# Betwixt and between

## Software in telecommunications and the programming language Chill, 1974 - 1999

by

Gard Paulsen

A dissertation submitted to BI Norwegian Business School
for the degree of PhD

PhD specialisation: Innovation and Entrepreneurship

Series of Dissertations 8/2011

**BI Norwegian Business School**

Gard Paulsen

Betwixt and between: Software in telecommunications and the programming language Chill, 1974 - 1999



Series of Dissertations 8/2011

ISBN: 978-82-8247-031-5
ISSN: 1502-2099

BI Norwegian Business School
N-0442 Oslo
Phone: +47 4641 0000
www.bi.no

Printing: Nordberg Trykk

The dissertation may be downloaded or ordered from our website
www.bi.no/en/Research/Research-Publications/

# Contents

**TABLE OF FIGURES** ..... V

**ACKNOWLEDGMENTS** ..... VI

**1. INTRODUCTION** ..... 1

HISTORIOGRAPHY ..... 7
THEORIES ..... 15
METHODS ..... 29
SOURCES ..... 33
OUTLINE ..... 37

**2. WHEN SWITCHES BECAME PROGRAMS: TELECOMMUNICATION AND COMPUTING, 1965 - 1974** ..... 39

PROGRAMMING LANGUAGES IN SCIENCE AND INDUSTRY ..... 40
SOFTWARE ENGINEERING ..... 47
PROGRAMMING SWITCHES ..... 50
INTERFACES AND INTERACTION ..... 54
CLOSER TO THE MACHINE ..... 57
COMPUTING AND THE INTERNATIONAL TELECOMMUNICATION REGIME ..... 62
STATE OF THE ART ..... 67
SOME CONCLUSIONS ..... 69

**3. CONTESTED DESIGNS: AGREEING ON A PROGRAMMING LANGUAGE FOR TELECOMMUNICATION** ..... 71

A TEAM OF SPECIALISTS? ..... 72
COORDINATING CONTESTATIONS ..... 80
MAINTAINING COMPATIBILITY ..... 85
ORCHESTRATING AGREEMENTS ..... 87
DESCRIPTIONS AND ALIGNMENTS ..... 89
POSTPONED DEADLINES AND DELAYED CONCEPTS ..... 92
THE STRUCTURE OF COLLABORATION ..... 96
SOME CONCLUSIONS ..... 103

**4. COMPROMISE AND COMPLEXITY: THE IMPLEMENTATION OF A PROGRAMMING LANGUAGE** ..... 107

DESIGN, IMPLEMENTATION AND FEEDBACKS ..... 108
IMPLEMENTING STRUCTURE ..... 113
COMPILING CHILL ..... 121
NORDIC COOPERATION AND COMPETITION ..... 124
COMPILATION IN THE ITT ..... 128
CONCURRENT PROCESSES AND DECISIONS ..... 130
FORMAL DEFINITION ..... 137
TRAVELLING IN CHILL ..... 141

"THERE NEED NOT BE A CONFLICT" – CHILL MEETS ADA ..... 143
SOME CONCLUSIONS ..... 149

**5. LARGE ORGANISATIONS AND LARGE SYSTEMS: THE USE OF CHILL IN LARGE TELECOMMUNICATION MANUFACTURERS ..... 151**

PROGRAMMING SYSTEMS ..... 152
CHILL IN THE ITT ..... 154
ENCOUNTERING C – CHILL, PHILIPS AND THE AT&T ..... 160
EARLY ADOPTERS AND EVOLUTIONS AT SIEMENS ..... 168
TAKING ON THE WORLD ..... 170
SOME CONCLUSIONS ..... 174

**6. ADVANCES AND REJECTIONS: ADMINISTRATIONS, COMMUNITIES AND THE STRUGGLE FOR DIFFUSION ..... 177**

COMMITMENTS ..... 177
AMBIGUITY AND NEGATIVITY ..... 181
BETRAYED FROM WITHIN ..... 183
MODULAR IMPROVEMENTS ..... 189
COORDINATED EMERGENCE ..... 194
THE CIRCULATION OF KNOWLEDGE ..... 201
CROSSING BOUNDARIES? ..... 206
SOME CONCLUSIONS ..... 208

**7. POSSIBILITIES AND OPPORTUNITIES: ENTERING MARKETS WITH CHILL ..... 211**

THE "ANCIEN REGIME" AND THE NEW BEGINNING ..... 211
NEW VENTURES AND NEW ENVIRONMENTS ..... 213
ENTREPRENEURSHIP, COMMUNITIES AND KNOWLEDGE ..... 218
SPINNING OFF ..... 221
ENTERING THE MARKET ..... 228
HARD TIMES ..... 234
THE VIABILITY OF INDEPENDENT VENDORS ..... 240
THE LONG POSTLUDE ..... 242
SOME CONCLUSIONS ..... 246

**8. CONCLUSIONS ..... 249**

THE DIRECTION OF TECHNOLOGICAL CHANGE ..... 252

**APPENDIX 1 ..... 259**

**APPENDIX 2 ..... 260**

**ARCHIVAL SOURCES ..... 263**

**INTERVIEWS ..... 264**

**PRINTED SOURCES ..... 265**

# Table of figures

Figure 2.1 Programming languages and the levels of abstractions.....42
Figure 2.2 Machine-oriented higher-level languages .....47
Figure 3.1 The Team of Specialists. .....79
Figure 3.2 The CCITT Study Group XI meeting in 1976.....95
Figure 3.3 Participants in the Team of Specialists.....101
Figure 3.4 The Team of Specialists, affiliations to meetings.....102
Figure 4.1 Phases of compilation.....111
Figure 4.2 Frequent participants in the Implementors' Forum. .....117
Figure 4.3 Participants and meetings in the Implementors' Forum. .....118
Figure 5.1 Temptations in research and realities of telecommunications...163
Figure 5.2 The bridge that almost was.....166
Figure 7.1 Chipsy as of 1993 .....230

Figure 3.1 is reproduced with the permission of Kristen Rekdal.
Figure 3.2 is reproduced with the permission of Remi Bourgonjon.
Figures 5.1 and 5.2 are reproduced with the permission of Kees Smedema.
Figure 7.1 is reproduced from *Telektronikk* 2/3 (1993), with permission of the article author, Kristen Rekdal.
Figures 2.1, 2.2 and 4.1 were created by Magnus Voll Mathiassen.

## Acknowledgments

For one who has spent most of his life listening to sounds, noises and music, it is a strange feeling handing in a thesis about a subject that is, in many respects, mute: neither history nor software reverberates much by itself, regardless of their perceived weight. Luckily, this thesis has not been made in silence or in an anechoic chamber. To those who have listened to my often confused ideas and questions, and to those who have given answers, advice and encouragement, I wish to express my gratitude. Without you, neither the history nor I would have been able to make much of an utterance.

This thesis has been supported by a scholarship from the BI Norwegian Business School. The department of Innovation and Economic Organisation has been a most stimulating working environment. All the members of the Center for Business History have provided me with inspiring discussions, insights and good spirits. The indefatigable Knut Sogner has supervised my work. His knowledge, encouragement and sheer work ethic have been incredibly motivating. Lars Thue, who has supported my work from the day I started out as a masters' student at the University of Oslo, has been an ever-present inspiration. My peer doctoral scholars in BI's doctoral programme have been great sources of encouragement and assistance.

Throughout the years, I have been helped by many. At the ITU archive in Geneva, I was given important help and advice from the head of the library and archive service, Kristine Clara. In Trondheim, Pål Tonstad Sandvik hosted me at the Norwegian University of Technology and Science (NTNU) when I was looking for sources. Svein Henrik Pedersen, also at the NTNU, voluntarily put boxes upon boxes of archival material in the back of his car and got them safely to Oslo, some 500 kilometres south. Also at the NTNU, the historian Ola Nordal helped me trace down lost files.

Throughout the process of research, I have received comments and advice from a number of eminent scholars. My loose association with the Tensions of Europe's SOFT-EU project provided me with numerous chances to receive comments from and listen to scholars with a keen interest in the history of software, often in great locations like Grenoble, Amsterdam and Lisbon. Gerard Alberts and David Nofre have been ever so kind to accommodate me in this group. At the annual conference of the Society of the History of Technology (SHOT), I have received important comments on very rough drafts from a number for commentators. Pascal Griset and Tom Haigh read a sketch of what became a chapter in this thesis. Their advice was invaluable. Peter B. Meyer, whom I also met at SHOT, was a great discussant and our long mail exchanges were always stimulating.

Long before this thesis found its subject, I received help from and was allowed generous interviews with Erick Chambe-Eng and Paul Olav Tvete, then of the Norwegian software firm Trolltech. While the subject of my

thesis led me astray from Trolltech, these early interviews were a huge inspiration. The many changes at Trolltech after these early interviews were conducted have also been an invaluable reality check for an historian, constantly reminding me of how quickly things change and how long a year can be, when experienced in real time.

Throughout the process of writing this thesis, I have received plenty of help related to writing, as I quickly found out that English was not as easy to write as I had first envisioned. At an early stage, Knut Kirknes and Marius Bakke corrected my faulty language to great effect. In the last round, Simon Niziol has brought the language up to scratch. His effort and professionalism have been very important and greatly appreciated. Erik Aadland, a peer doctoral fellow at BI, helped me out when I had to grasp a completely different language, namely that of statistics and structured equation modelling. I would have been lost without his help.

Most importantly, many participants in the history I have looked into have been very helpful and generous with their time. Many hours have been spent in conversations with interesting and knowledgeable people, both in Norway, in the Netherlands and over the wires. All help, answers and comments have been greatly appreciated. Most importantly was the generosity shown by Kristen Rekdal. Without his collection of papers, documents and communications, this thesis would have been something completely different. The trust he showed me when allowing me unconditional access to this material will remain unsurpassed.

The care, encouragement and patience showed by my friends, family and loved ones have meant the world to me. Just like the sounds, noises and music of the everyday, I could not have survived without them.

# 1. Introduction

This thesis studies the creation, use and ultimate demise of a rather peculiar high-level programming language named Chill.[1] It was peculiar in its origin, a United Nations specialised agency. It was peculiar in its application area, which was programming of large-scale telecommunication switches. It was also peculiar in its process of realisation, which was done within an international committee, consisting of a number of computer scientists, telecommunication experts and bureaucrats from different organisations and countries. The negotiations went on inside the committee for almost six years before the language was unleashed in 1980, as an official recommendation of the International Telecommunication Union (ITU).[2]

By 1990, Chill was the only programming language that was common to more than one of the major public telecom switching systems that were in use. By that time, more than 12,000 programmers had used the language in one way or another.[3] 20 years after its inception, by the late 1990s, Chill was a marginal technology. It was almost close to extinction. No new software developments were made with the language and by 1999 the ITU published what was to be the last maintained version of Chill.[4] Still, legacy Chill code lives on in telecommunication systems that continue to run today. This thesis explains how and why the telecommunication industry first handed over the responsibility for a key technology to a group of programming language designers, and then how it readily would apply the results, only to abandon the technology a few years later.

---

[1] CHILL is an acronym for CCITT High Level Language. CCITT was, in turn, an abbreviation for Comité Consultatif International Télégraphique et Téléphonique (the International Telegraph and Telephone Consultative Committee), which was the technical wing of the International Telecommunication Union (ITU). CCITT is now the ITU's Telecommunication Standardization Sector (ITU-T), but I will use the term CCITT throughout this thesis. For reasons of readability, I will not capitalise the name Chill in the running text. This follows the typographical conventions for the rest of the thesis: names that are more or less pronounceable are treated as proper names and written as 'Chill', whereas unpronounceable acronyms are written in a capitalised form. In direct quotation, however, the style adopted by the original source is preserved. An example of Chill code is given in Appendix 1.

[2] *CCITT High Level Language (CHILL),* CCITT Recommendation Z.200*,* (1980).

[3] Chill was used in some of the most successful switches on the market, like the System 12 by the ITT (later Alcatel) and the EWSD switches by Siemens. On the use and status of Chill in the early 1990s, see Kristen Rekdal, "CHILL - The International Standard Language for Telecommunications Programming", *Telektronikk,* 89, no. 2/3 (1993).

[4] *CHILL - The ITU-T Programming Language*, ITU-T Recommendation Z.200 (1999).

The peculiar programming language Chill was neither a total failure nor a total success. This in-between status was reflected in its features, its background and its process of realisation. Technologically, Chill was a programming language that was one out of many. It tried to reconcile the particular needs of real-time communication systems with the generality of high-level programming language principles, an aim shared by many other programming languages designed in the 1970s.[5] It also extended on the design practices common to programming languages of an older vintage, like the pioneering high-level programming languages of the 1960s. On an even more general level, Chill was part of a larger shift towards programming as a dominant activity in the telecommunication industry, where balls of wires were replaced by loops of programming code.[6] Historically, it was created within the vicinity of what was perceived as the anchor of an oligopolistic regime of telecommunication administrations and manufacturers, the ITU, right before a comprehensive organisational transformation of the telecommunication industry. Organisationally, it was created by a committee torn between the agendas of several communities of technological practitioners, telecommunication administrations and manufacturers. It was really "betwixt and between".[7]

This thesis studies this peculiar technology, from its inception in the first half of the 1970s and up until the last maintained publication of the standard by the ITU in 1999.[8] I approach this through a detailed study of how Chill was shaped during its life cycle. I explain which priorities gained

[5] A comparison could be the programming language Ada, which was commissioned by the American Department of Defence in the mid-1970s. For an overview, see William A. Whitaker, "Ada—the project: the DoD high order language working group", *ACM SIGPLAN Notices* 28, no. 3 (1993).

[6] A contemporaneous review is M. T. Hills and S. Kano, *Programming electronic switching systems - real-time aspects and their language implications*, IEE Telecommunications Series (Stevenage: Peter Peregrinus Ltd, 1976).

[7] I have appropriated the term "betwixt and between" from a classic essay by the anthropologist Victor Turner, which was concerned with initiation rituals and transition ceremonies from one social status to another. My use of Turner's expression is intended as an illustration of transition periods of a rather different kind than those of initiation and social transition and carries no further empirical or theoretical denotations to Turner's work. On the direct meaning of the expression, see Victor Witter Turner, *The forest of symbols; aspects of Ndembu ritual* (Ithaca, N.Y.: Cornell University Press, 1967), 93-111.

[8] While long-term studies of the industrial structure as a consequence of life-cycle developments have flourished, few studies of a specific technology life cycle have been published. On this problem, and an effort to somewhat rectify this, see William Walker, "Entrapment in large technology systems: institutional commitment and power relations", *Research Policy* 29, no. 7-8 (2000).

prominence in which period, by which mechanisms and who carried them through. Furthermore, I consider how the changing political economy of the telecommunication industry and the strategies of administrations and manufacturers shaped the fate of the language. Together, the technical features and the use of the language are understood as the direction of technological change that was constituted by Chill. In particular, I am interested in the move towards a common and standardised hybrid high-level programming language: general, yet specialised, high level, yet efficient, common, yet atypical.

I approach the question of the direction of technological change through an analysis of the technical diplomacy throughout the phases of emergence, use and demise. This diplomacy was related to both its technical features and its use. The diplomacy happened both on the level of quarrelling over language concepts as well as on an organisational level about how binding the supporting organisation saw their commitment to Chill. This approach to technological change, as a diplomatic process, combines research into decision-making at the individual level, negotiations at community level, firm-level strategising and the role of institutional regimes. I particularly look into how shared norms and ideals held in communities of technological practitioners shaped Chill and how the changing political economy of telecommunications intervened in its life.[9] I also investigate how different strategies on the division of programming labour among telecommunication administrations and manufacturers influenced the design and use of the programming language. This makes it possible to analyse processually how, to what extent and in which periods the various sources of influence dominated the Chill life cycle.

The main period under investigation, from the early 1970s to around 1990, was one of substantive technical change and an emerging organisational transformation of the telecommunications field. The introduction of digital transmission and computer-controlled switching transformed the telecommunication infrastructures dramatically over the course of two decades, moving the industry from the analogue to the digital domain. The organisational principles of telecommunications were also about to undergo radical changes, as the dominant pattern of a strongly regulated industry was put under increasing pressure, although the real liberalisation of the industry was still some years away. This coincided with what has been proposed to be a general shift in business, where the role of

[9] The term "communities of technological practitioners" builds on Edward W. Constant, *The origins of the turbojet revolution*, Johns Hopkins studies in the history of technology (Baltimore: Johns Hopkins University Press, 1980). A more refined definition and clarification regarding similar concepts is developed in a later section of this chapter.

the large multidivisional firm as a generator of innovation and growth diminished, a change in the direction of "deverticalisation".[10] Furthermore, it concurred with the emergence of what has been described as the "knowledge-based economy", a term used to describe the centrality of science and technology within sectors such as pharmaceuticals and information and communication technologies, and the term has increasingly been used to describe economies where the importance of the information sectors is high and the "share of intangible capital is greater than that of tangible capital in the overall stock of real capital".[11] However, the 1970s and early 1980s were also something in between. It was a period of transition between the numbing stability before the 1970s and the raucous revolutions that turned the industry upside down in the 1990s. The period was more like what Victor Turner found to be a common phase in initiation rituals, a "liminal period" where things were "betwixt-and-between", both technically and organisationally.[12]

To those involved in the development of software for telecommunication systems at the time, the 1970s and early 1980s were also a period of "fruitful darkness", where "king and people are closely identified", to paraphrase Victor Turner's anthropology once again.[13] In this darkness, decisions about new and novel technologies like programming languages were of a different kind than in projects of "normal engineering" or of radical inventive development. The practitioners and scientists involved in the Chill project knew a lot about programming languages before designing one for telecommunication systems. This was based on prior experience in general computing – but the combination of computing and telecommunications was still uncharted territory, unknown and untested.[14] It was somewhat risky, but not completely uncertain, made by daring

[10] Richard N. Langlois, "The vanishing hand: the changing dynamics of industrial capitalism", *Industrial and Corporate Change* 12, no. 2 (2003).
[11] Dominique Foray, *Economics of knowledge* (Cambridge, Mass.: MIT Press, 2004), ix. Foray, *Economics of knowledge*; Robin Cowan, Paul A. David, and Dominique Foray, "The Explicit Economics of Knowledge Codification and Tacitness", *Industrial and Corporate Change* 9, no. 2 (2000). For critical discussions, see Richard N. Langlois, "Knowledge, Consumption, and Endogenous Growth", *Journal of Evolutionary Economics* 11, no. 1 (2001).
[12] Turner, *The forest of symbols; aspects of Ndembu ritual*.
[13] Ibid., 110
[14] The foundations of programming language design were laid in the late 1950s and 1960s. In 1969, it was already time to release a tome on the history of programming languages. See Jean E. Sammet, *Programming languages: history and fundamentals*, Prentice-Hall series in automatic computation (Englewood Cliffs, N.J.,: Prentice-Hall, 1969).

technological practitioners rather than heroic inventors.[15] As in the rituals investigated by Turner, technological change under liminality can be characterised by a duality, where the process was ambiguous but still goal-directed at its outset and depended on successful reintegration into the economic and technical system at its end. This thesis analyses the combination of telecommunication knowledge and computer knowledge and explicates how individuals, communities and organisations acted under liminality and how they tried to reintegrate the combinatory knowledge into stable arrangements.

Chill originated in the early 1970s when the ITU drummed together experts from the telecommunication industry and the computer field, and put them to work on solving the mounting difficulties related to programming telecommunication equipment. The prospect of one common programming language, a technology that could be shared between manufacturers and administrations, gathered support. Several large manufacturers, like the multinational firm ITT, the Swedish company L. M. Ericsson, Siemens of Germany and the Dutch firm Philips, put their weight behind the proposal early on. During the design process, the technologically leading company, the American AT&T, participated through its research branch Bell Laboratories. European administrations backed the initiative from the outset and the Japanese NTT followed suit. The organisations' decisions to participate in the Chill project were rooted in different agendas, in particular about who should control the programming of new telecommunication equipment, but also in strategies of international expansion and exports.

In retrospect, the fate of Chill looks almost inevitable. Why would anyone in their right mind design a programming language especially for the application domain of telecommunication switching, from the ground up, in an international committee, right before the comprehensive organisational transformation of the telecommunication industry? According to Remi Bourgonjon, who led the committee in the ITU that was responsible for Chill, a common programming language was perhaps not the best of ideas: "In hindsight it was totally stupid," he said when I interviewed him.[16] However, at the time the project made sense to the participants involved in the project and to its sponsors. "It made perfect sense," Bourgonjon told me.[17] What apparently looked like a rather bad idea in retrospect was also part of a general trend, as projects with similar technical aims and

[15] On risk and uncertainty, see Frank H. Knight, *Risk, uncertainty and profit* (Boston and New York,1921).
[16] Remi Bourgonjon, interview with author, 16 January 2009, Heeze, the Netherlands.
[17] Ibid.

organisational background were initiated in comparable industries a few years both before and after the Chill initiative.[18]

An historical understanding of Chill's fate must be rooted in the context of the project itself, how it was understood by its participants and contemporaries and how it was related to the actions and strategies of telecommunication manufacturers, administrations and international organisations at the time. A proper account of the creation, use and demise of the language as it appeared at the time will allow an historical understanding of the technology. Subsequently I analyse whether what can appear as failures to us now was a result of decisions made within the project or shaped by factors external to the participants control and understanding. This might also contribute towards a better understanding of technical change in software development in telecommunications in general.

This thesis also has a general ambition: it aims at explaining how and why the telecommunication industry started to use high-level programming languages in the development and production of telecommunication equipment. It seeks out an understanding of the direction of this technical change and its organisational underpinnings and results. There are three reasons for this ambition. Firstly, the empirical context of international cooperation and transnational collaboration makes it possible to understand how technological choices were made through technical diplomacy at a level beyond local circumstance. By accounting for the international context of technological change, it is possible to analyse the general conditions and priorities that directed the particular evolution of Chill. Secondly, the unruly and peculiar nature of Chill encourages such general ambitions: Chill involved, in some way or another, almost all large telecommunication manufacturers and most telecommunication administrations of importance in the period and allows a detailed understanding of how programming with specialised, yet high-level, languages was perceived as a viable route in the sector at large. The sources available in the study of this *one* programming language highlight the development and use of programming languages in telecommunication at large. The third reason for this general ambition is theoretical: this thesis tries to search out the limits of established models of product life cycles at both the particular and the product group level of programming languages for telecommunication systems. This necessitates a

[18] The US Department of Defense embarked on a very similar project when they tried to standardise the programming language Ada in many of their operations from the late 1970s. See Whitaker, "Ada—the project: the DoD high order language working group". Before that, similar projects were initiated and standardised in the technical field of process control. See I. D. Hill and B. L. Meek, *Programming language standardisation*, Ellis Horwood series in computers and their applications (Chichester, Eng., New York: E. Horwood ; Halsted Press, 1980).

general understanding of the direction of the technical change that the turn to high-level programming languages constituted in telecommunications, and an understanding of through what means this turn was sought.

## Historiography

Chill was a programming language similar to many others. It shared both technological and organisational similarities to other so-called real-time, parallel or concurrent programming languages. Such similarities can only be understood if we approach the subject in a manner that steps beyond local circumstance and the peculiarities of Chill. Consequently, I approach the development of Chill and high-level programming languages in a different way than the one that dominates in the history and sociology of technology. In this thesis, there is greater focus on international cooperation and local circumstance is less in the foreground than is usual.[19] Causes of the technological change and the similarities between high-level languages of the time are sought at the level of international communities of technological practitioners as well as embedded in the strategies of telecommunication administrations, equipment manufacturers and research establishments. Ultimately, these interests were reconciled at the level of an international organisation, bound together in what can be understood as the international telecommunication regime. This moves the thesis away from the internalistic approaches that have dominated the history of programming languages for a long time, where the development and design of such technologies often is considered as solely an intellectual undertaking rooted in the academic discipline of computer science, or as responses to needs caused by advances

[19] The view that science and technology are predominantly about local and contingent practices is a claim that permeates much of the so-called constructivist writing on science and technology. See, for example, many of the contributions in Edward J. Hackett et al., *The Handbook of Science and Technology Studies, Third Edition* (Cambridge, Mass.: The MIT Press, 2007). It would be unfair to accuse the large body of work of constructivist writing as completely ignoring the international level. See, for example, how issues of locality and international networks are integrated in David Wade Chambers and Richard Gillespie, "Locality in the History of Science: Colonial Science, Technoscience, and Indigenous Knowledge", *Osiris* 15(2000). Furthermore, a turn towards the transnational has recently emerged also in the history of technology, in a move that resonates well with tendencies in general history. See for example Alexander Badenoch and Andreas Fickers, *Materializing Europe : transnational infrastructures and the project of Europe* (New York: Palgrave Macmillan, 2010).

in computer hardware.[20] Still, the thesis aims at a goal common to much of this literature, as it explicitly tries to open up the black box of technology and render what is typically impermeable more or less transparent.[21]

Earlier research on the history of Chill can be accused of being kept well within a black box, although the available publications are not without strengths. In the Robert J. Chapuis and Joel E. Amos tome on telephone switching technology from 1960 to 1985, one of a few publications where the history of Chill is analysed by someone outside the Chill project, the technical details of the programming language are kept to a minimum.[22] Chill is instead briefly presented as an efficient technology, but also as a standard that enjoyed "relatively limited spread of use".[23] However, as the chapter was published in 1990, Chapuis and Joel argue that more widespread use of Chill *might* happen as a consequence of the radical concentration of the switching industry that was observed at that time, while the limited appeal of the language up until 1990 was generally explained by the popularity of one of Chill's competitors, the programming language C. Chapuis and Joel place the development of Chill in an international framework, presenting the programming language as a part of ITU's move towards the standardisation of other computer-related standards.[24]

---

[20] On the internalistic bent in the historiography of programming languages, see Jan Rune Holmevik, *Educating the machine : a study in the history of computing and the construction of the SIMULA programming language*, STS rapport ; nr 22 (Dragvoll: Senter for teknologi og samfunn, Universitetet i Trondheim, 1994); Peter Mark Priestley, "Logic and the development of programming languages, 1930 - 1975" (University College London, 2008). The literature in question is largely available through the proceedings of the three "history of programming languages" conferences (HOPL) held in 1981, 1996 and 2007.

[21] The black box metaphor has been persistently used in both economic and sociological approaches to technology. See Nathan Rosenberg, *Inside the black box : technology and economics* (Cambridge [Cambridgeshire] ; New York: Cambridge University Press, 1982); Bruno Latour, *Science in action : how to follow scientists and engineers through society* (Cambridge, Mass.: Harvard University Press, 1987).

[22] See Robert J. Chapuis and Amos E. Joel, *Electronics, computers and telephone switching: 1960-1985*, 2 vols., vol. 2, Studies in telecommunication (1990), 267-90; Kristen Rekdal, "CHILL - The Standard language for Programming SPC Systems", *IEEE Transactions on Communications* 30, no. 6 (1982); Rekdal, "CHILL - The International Standard Language for Telecommunications Programming"; C. H. Smedema, "Some Issues in the International Standardization of CHILL and Ada", *Computers & Standards* 4, no. 2 (1985).

[23] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 283.

[24] This is no coincidence, as Robert Chapuis was very much part of the CCITT community. On his background, see *Robert Chapuis*, an oral history conducted in 1993 by Frederik Nebeker, IEEE History Center, New Brunswick, NJ, USA.

The opposite of this international outlook is found in a comprehensive historical investigation of the computer department of the Norwegian University of Science and Technology, where the historian Ola Nordal explicates the Norwegian setting of Chill in some detail, highlighting the particular organisational framework of the Norwegian contributions to the CCITT project.[25] The way Chill is presented here, as a part of a trajectory of programming language research at one particular organisation, makes Nordal's analysis part of a local history, and consequently presents only a part of the Chill history. Still, this is one of the few publications that present the wider implications of Chill for Norwegian industry, as it studies a Norwegian effort to commercialise knowledge obtained through the Norwegian participation in the Chill project as part of a broader study of spin-offs and commercialisations from the technical university of Norway related to computing.

Some of the participants in the Chill project have also published brief overviews of the history of Chill.[26] The general view put forward in these is that Chill was a viable language with some particular technological and organisational strengths. While these publications offer some good overviews of how Chill was realised, none of them follows the technology up until its end.

On the history of the ITU, Chill's organisational origin, a more comprehensive literature exists that highlights the development of the organisation.[27] The ITU was created in 1932 by joining the International Telegraph Union (founded in 1865) and the signatories of the International Radio Telegraph Convention of 1906. By the late 1960s, the main bodies of

---

[25] See Ola Nordal, *Verktøy og vitenskap: datahistorien ved NTNU* (Trondheim: Tapir akademisk, 2010), 172-78.

[26] For some noteworthy examples of work by insiders from the Chill project, see Rekdal, "CHILL - The International Standard Language for Telecommunications Programming"; Smedema, "Some Issues in the International Standardization of CHILL and Ada"; C. H. Smedema, P. Medema, and M. Boasson, *The programming languages : Pascal, Modula, CHILL, and Ada* (Englewood Cliffs, N.J.: Prentice/Hall International, 1983).

[27] The standard history of the ITU is Harold K. Jacobson, "ITU: A potpurri of Bureaucrats and Industrialists", in *The Anatomy of Influence*, ed. Robert Cox and Harold K. Jacobson (New Haven, Conn.: Yale University Press, 1973). See also George A. Codding and Anthony M. Rutkowski, *The International Telecommunication Union in a changing world* (Dedham, MA: Artech House, 1982); George A. Codding, *The International Telecommunication Union; an experiment in international cooperation* (Leiden,: E. J. Brill, 1952). For a more recent overview, see William J. Drake, "The Rise and Decline of the International Telecommunications Regime", in *Regulating the Global Information Society*, ed. Christopher T. Marsden (London: Routledge, 2000).

the ITU were the International Consultative Committee for Telegraph and Telephones (CCITT), the International Consultative Committee for Radio (CCIR) and the International Frequency Registration Board. These bodies were responsible for developing recommendations about telecommunications standards, developing telecommunications facilities and networks, establishing the lowest possible rates consistent with efficient service, allocating the radio frequency spectrum, registering radio frequency assignments, coordinating orbital slots for communications satellites and helping developing countries to improve their telecommunications equipment and networks. The system was based on a one vote, one nation principle, basically meaning the vote of the national telecommunication administration, which often held monopolistic power over telecommunication services in this period. A few studies have successfully engaged in analysing the regulatory regime of international telecommunications within the political sciences and sociology. [28] In particular, Peter Cowhey has claimed there was a strong relationship between the national monopolies and the international telecommunication organisation, and has claimed that the ITU and the CCITT sustained "one of the most lucrative and technologically significant international cartels in history".[29] According to Cowhey, the CCITT was "the anchor of a regime that facilitated bilateral monopolistic bargains, reinforced national monopolies, and limited the rights of private firms in the global market".[30] To Cowhey, the CCITT was also an "epistemic community devoted to the idea of a 'natural monopoly' for telephone services".[31] This community would pervert technologies into something that would be beneficial for the established regime. To the CCITT veteran Gerd Wallenstein, the CCITT was something completely different. It was "a transnational subculture held together by technical expertise and a specialised language of their own making".[32] It was a suprapower in the transnational structure of standards, although fraught with infighting and difficulties. Its mission was the creation

[28] See Drake, "The Rise and Decline of the International Telecommunications Regime"; Peter F. Cowhey, "The international telecommunications regime: the political roots of regimes for high technology", *International Organization* 44, no. 169-199 (1990); Eli Noam, "International Telecommunications in Transition", in *Changing the Rules: Technological Change, International Competition, and Regulation in Communications*, ed. Robert W. Crandall and Kenneth Flamm (Washington, D.C.: The Brookings Institution, 1989).

[29] Cowhey, "The international telecommunications regime: the political roots of regimes for high technology".

[30] Ibid.: 176.

[31] Ibid.: 173.

[32] Gerd Wallenstein, *Setting Global Telecommunication Standards: The Stakes, The Players & The Process* (Norwood, MA: Artech House, 1990), 1.

of non-binding technical recommendation through "mutually distrustful bedfellows", which Wallerstein believed fostered standards and technologies that works as the law in civil law countries, "presumably fair, equitable, and common property of mankind".[33] To Wallerstein, the system was a safeguard against the problems that Cowhey ascribed to it.

Few of these studies have been concerned with technological development, but remained focused on regulatory issues. Furthermore, much of this work has seldom been concerned with the actor level and how individuals act within the boundaries of regimes and organisations.[34] The institutional orientation common to much of this research has rendered the individuals active in the international telecommunication organisations as captive to the national stratified logic of the industry. In this thesis, I analyse how regime logics and community norms are interrelated in some periods and opposed to each other in other periods. As such, I develop an historical understanding of regimes and community norms that goes beyond the simple handcuffs of the international telecommunication regime of Cowhey, without succumbing to the technocratic rule envisioned by Wallerstein.

As part of the history of software in general and programming languages in particular, the history of Chill adds to the available literature in several ways: at the outset of the Chill project, some three-quarters of the productive energies of the computer industry were going into software.[35] Nevertheless, the historical literature has been slow in recognising this point. What is more, the specialised history of computing has been slow in recognising the importance of software altogether.[36] Historical explorations of the software industry and professions have been published, and a recent emphasis on applications and the societal changes resulting from software

[33] Ibid.

[34] One notable exception to both concerns is Susanne K. Schmidt and Raymund Werle, *Coordinating technology : studies in the international standardization of telecommunications*, Inside technology (Cambridge, Mass.: MIT Press, 1998).

[35] Barry W. Boehm, "Software and its Impact: A Quantitative Assessment", *Datamation* 19, no. 5 (1973). Here referred from Michael S. Mahoney, "Software: The Self-Programming Machine", in *From 0 to 1: An Authoritative History of Modern Computing*, ed. Atsushi Akera and Frederik Nebeker (Oxford: Oxford University Press, 2002).

[36] Mahoney, "Software: The Self-Programming Machine", 92.

are slowly emerging.[37] Some important contributions to the history of software have also integrated this into research on more general historical interest. One example is found in Atsushi Akera's book *Calculating a natural World*.[38] Another example is the historically oriented sociologist Donald MacKenzie's book *Mechanizing Proof*, in which through research into the history of software the author makes a significant contribution to the sociology of knowledge.[39] My thesis tries to follow suit, although the general context is that of international cooperation and collaboration in the 20th century.[40]

The rather specialised field of the history of programming languages has been one of the best-covered themes within the history of computing. This literature has, however, remained largely internalistic and has largely been written by specialists in the field.[41] A typical tendency is the way they have imposed a logic and coherence that was typically absent at the time,

---

[37] A more recent historiographical overview is Martin Campbell-Kelly, "The History of the History of Software", *IEEE Annals of the History of Computing* 29, no. 4 (2007). On the history of the software industry, see ———, *From airline reservations to Sonic the Hedgehog : a history of the software industry*, History of computing (Cambridge, Mass.: MIT Press, 2003). On the use of software, see JoAnne Yates, *Structuring the information age : life insurance and technology in the twentieth century*, Studies in industry and society (Baltimore, Md.: Johns Hopkins University Press, 2005). On the professions of software workers, see Nathan Ensmenger, *The computer boys take over : computers, programmers, and the politics of technical expertise*, History of computing (Cambridge, Mass.: The MIT Press, 2010).. See also Thomas Haigh, "How Data Got its Base: Information Storage Software in the 1950s and 1960s", *IEEE Annals of the History of Computing* 2009.

[38] Atsushi Akera, *Calculating a natural world : scientists, engineers, and computers during the rise of U.S. cold war research*, Inside technology (Cambridge, Mass.: MIT Press, 2007).

[39] Donald A. MacKenzie, *Mechanizing proof : computing, risk, and trust*, Inside technology (Cambridge, Mass.: MIT Press, 2001).

[40] On the historical role of international organisations in general, see Akira Iriye, *Global Community: The Role of International Organizations in the Making of the Contemporary World* (Berkeley: University of California Press, 2002). International organisations and relations have also attracted interest within the history of science and technology, for a recent example, see John Krige and Kai-Henrik Barth, *Global power knowledge: science and technology in international affairs* (Chicago, Ill.: University of Chicago Press, 2006).

[41] Sammet, *Programming languages: history and fundamentals*; Richard L. Wexelblat, *History of programming languages* (New York: Academic Press, 1981); Thomas J. Bergin and Richard G. Gibson, *History of programming languages II* (New York: ACM Press; Addison-Wesley Pub. Co., 1996); Thomas J. Bergin, "A history of the history of programming languages ", *Communications of the ACM* 50, no. 5 (2007).

and that they render the evolution of programming languages as a genealogy separate from its context. This does not, however, mean that this internalistic body of work is without strengths. A particularly important contribution is a recent study on the relationship between software engineering and programming language design, which has shown how various application domains and the research field of software engineering shaped programming language design in the 1970s.[42] Empirically, this is closely related to the study of the design and use of programming languages in telecommunications. However, a particular emphasis in terms of theoretical orientation or dominant contextualisation is difficult to find. Two notable exceptions are Jan Rune Holmevik's study of the programming language Simula and Mark Priestley's study of the role of logic in what he calls the Algol research programme.[43] Both approach the history of programming languages with an ambition of going beyond the internalistic tendencies. Where Priestley grounds his study in theoretical approaches common to the traditional history of science, Holmevik's study reflects the wider history of research funding, research politics and the understanding of the computer in Norwegian society.

The business history of the software industry has been engaged with an economic approach to software, although it has seldom been engaged with programming languages. This is primarily evident in the books and articles of Martin Campbell-Kelly.[44] In his survey of the history of the software industry, *From airline reservation to Sonic the Hedgehog*, he highlights the dynamics of the software industry from its early days and up until the mid-1990s. Campbell-Kelly is explicitly focusing on the software industry as a whole, while I am concentrating of a part of the industry that existed on the fringes of both telecommunications and computing.

---

[42] Barbara G. Ryder, Mary Lou Soffa, and Margaret Burnett, "The Impact of Software Engineering Research on Modern Programming Languages", *ACM Transactions on Software Engineering and Methodology* 14, no. 4 (2005).

[43] Priestley, "Logic and the development of programming languages, 1930 - 1975"; Holmevik, *Educating the machine : a study in the history of computing and the construction of the SIMULA programming language*; Jan Rune Holmevik, *Inside innovation: The Simula Research Laboratory and the History of the Simula Programming Language* (Oslo: Simula Research Laboratory, 2004).

[44] Most prominently, Campbell-Kelly, *From airline reservations to Sonic the Hedgehog : a history of the software industry*. For an examination of more recent trends, see Martin Campbell-Kelly and Daniel D. Garcia-Swartz, "From Products to Services: The Software Industry in the Internet Era", *Business History Review* 81, no. Winter 2007 (2007). Another effort is found within Knut Sogner, *En liten brikke i et stort spill : den norske IT-industrien fra krise til vekst 1975-2000* (Bergen: Fagbokforl., 2002).

The role of innovation in software production and development has been analysed by scholars working on a different level of analysis than Campbell-Kelly, namely within the "national innovation systems" approach pioneered by Christopher Freeman, Richard Nelson, and Bengt-Åke Lundvall.[45] Two different studies, conducted more or less within this approach, has approached the software sector in general: David Mowery has compared the development of the U.S and Japanese software industries, where a number of systemic differences such as financial systems and intellectual property rights regimes are used as explanatory factors of different development paths. Mowery's study is relevant to understand the industrial dynamics of new entrants in the software industry, but is not concerned with software as an activity of other industries, such as telecommunications.[46] Furthermore, Mowery highlights national characteristics and national innovation systems, rather than the transnational and international character of the knowledge communities examined in this project. Secondly, Ed Steinmueller has written two historically oriented articles on the American and the European software industries.[47] According to Steinmueller, "the sites of knowledge generation in the software industry are extremely dispersed among disciplines and organisations". [48] Knowledge is gained through imitation and experimentation, in problem-solving related to specific and situated bottlenecks or innovative ideas, but also from basic research into computer science and software engineering.[49] This dispersed nature is readily present in the history of software development in telecommunications.

The existing history of Chill, software development and programming languages can be summarised as partly too internalistic, and while the general literature has recently turned towards issues such as how external factors can shape emergence and use, it has only to a limited extent been interested in economic aspects of this history. The considerable literature on the evolution of telecommunications, on the other hand, abounds with

---

[45] For an overview of the literature, see Jan Fagerberg, "Innovation: A guide to the literature", in *The Oxford Handbook of Innovation*, ed. Jan Fagerberg, David C. Mowery, and Richard Nelson (Oxford: Oxford University Press, 2005).

[46] David C. Mowery, *The international computer software industry : a comparative study of industry evolution and structure* (New York: Oxford University Press, 1996).

[47] W. Edward Steinmueller, "The US Software Industry: An Analysis and Interpretive History", in *International Computer Software Industry*, ed. David C. Mowery (Oxford: Oxford University Press, 1996); ———, "The European software sectoral system of innovation", in *Sectoral Systems of Innovation: Concepts, Issues and Analyses of Six Major Sectors in Europe*, ed. Franco Malerba (Cambridge: Cambridge University Press, 2004).

[48] Steinmueller, "The European software sectoral system of innovation", 221.

[49] Ibid.

research into organisational aspects, almost to an extent that other issues have been overlooked. In particular, the relationship between the institutional setting of the industry and the direction of technological change under its control has received little attention. Furthermore, parts of this literature are too nationally oriented, in particular when concerned with technology, and those few concerned with international cooperation have largely regarded this level as captive to the national monopolistic systems, and consequently give little agency to the participants in projects such as Chill.

## Theories

My analysis of Chill is grounded in some general conceptualisations and theoretical assumptions. Below, I provide some clarifications and definitions regarding terminology. Following this, I discuss how my application of these concepts is related to larger theoretical concerns.

Changes in programming technologies are cases of *technical or technological change*, and should be grounded in established theoretical conceptualisations of this.[50] Here, it makes sense to understand *change* in the broadest possible sense, as something that necessitates novel ideas and the application of them, which makes it comparable to the commonly held distinction between invention and innovation, although here I restrict it to the realm of technology.[51] Technological change, consequently, is something that implies use, as there has simply not been a change in technology if it has not been put into use.

The term *technological change* has, at least throughout the last century, been given a meaning that predominantly includes changes in knowledge, practices and artefacts, the word *technical change* has often been confined to more narrowly defined works within economics.[52] That term has first and foremost been used to describe situations where firms choose

[50] For a philosophical exposition on roughly the same issues, see Jon Elster, *Explaining Technical Change*, ed. Jon Elster and Gudmund Hernes, Studies in Rationality and Social Change (Cambridge: Cambridge University Press, 1983).

[51] The distinction between invention and innovation is typically attributed to the influential economist Joseph Schumpeter. See in particular chapters three and four in Joseph Alois Schumpeter, *Business cycles; a theoretical, historical, and statistical analysis of the capitalist process*, 1st ed. (New York, London,: McGraw-Hill Book Company, inc., 1939). For a discussion, see Vernon W. Ruttan, "Usher and Schumpeter on Invention, Innovation, and Technological Change", *The Quarterly Journal of Economics* 73, no. 4 (1959).

[52] On the changing meaning of the word technology, see Eric Schatzenberg, "Technik Comes to America: Changing Meanings of Technology before 1930", *Technology and Culture* 47, no. 3 (2006); Leo Marx, "Technology - the Emergence of a Hazardous Concept", *Technology and Culture* 51, no. 3 (2010).

different techniques when producing a given output. In the following, I will use the two terms more or less interchangeably, where technological change includes, but is not restricted to, technical change in the production of telecommunication equipment. The term *direction* of technological change has traditionally been confined to the same constituencies of economic research that limit the term technical change to choice of techniques producing a given output. Here, the *direction* of the change has been restricted to the factor bias, typically restricted to whether it saves on labour, capital or energy.[53] In the following, I apply the term direction to understand the technological specificities of programming language design *and* use, where it has been typical to argue that the technology evolved towards higher levels of abstractions and degrees of modularity through time. Other directional concepts, like increased reliability and testability of software, should also be considered part of this broader concept of the direction of technological change. Chill was a change towards a relatively high-level hybrid language, especially designed for telecommunication systems, and intended as a common and standardised language.

The term direction is thus applied to highlight choices, which implies that by other design decisions the technological change would have looked different. Design should here, in the way proposed by Edward Layton, be understood as the purposeful and value-laden application of technological means, integrating technological knowledge with social and economic aims.[54] Use is understood as intertwined with design decisions and equally purposeful. Chill was a language designed to reduce variety, as a standard, and it opted for a technical solution that can be described as a hybrid, catering towards high levels of abstractions as well as the computing efficiency of lower level code.

I approach the fate of Chill through its full *life cycle*. Still, it has been important to step away from the rigid conceptualisation of product life cycles, where innovation happens in one distinct period of a technology's

[53] Elster, *Explaining Technical Change*.
[54] Edward T. Layton, "Technology as Knowledge", *Technology and Culture* 15, no. 1 (1974).

life.[55] In particular, I have highlighted how innovation was made possible as well as limited by decisions influenced by the community of technological practitioners and by the strategising among interconnected firms and organisations in all three phases of life: birth, maturity and death. Still, the phase of birth was the main design phase of Chill, from around 1974 to 1980. I will typically refer to this phase as *emergence*. The period where the language diffused and was used equals the period of maturity, roughly spanning the years between 1980 and 1990. I often refer to these years as the years of *diffusion* or *use*. The years where the language was more or less confined to legacy systems, from 1991 to 1999, I refer to as the years of *demise*.

In many ways, Chill was not a typical technology understood as an artefact. Chill was also an effort of creating a legislative binding agreement on how a programming language for telecommunication equipment should be and by whom it should be used, embodied as a text – almost a legal document. Its fate was very much dependent on how this recommendation was received and how it was put into action. As a whole, the process was an exercise of standardisation, a sub-case of *institutionalisation*.[56] Furthermore, programming languages can be regarded as institutionalisation per se, as they always involve ways of codifying programming techniques into a set of rules and procedures, rules and procedures that can be tools when programming – which is the act of creating programs. Consequently, it

[55] The way I use the term ”life cycle” is different from the product life cycle concept common in some quarters of economics and marketing. Where the product life cycle concept of economics has mainly been concerned with the product group level and the industrial structure underpinning it, I am solely focused on the changes over the course of one technology’s life cycle. However, I do adopt the three-pronged characterisation of technology development – birth, maturity, death (without any claims about the natural passing of such periods) – but hold the questions on innovation, shaping and standardisation of the product more open than what is common in the “conventional model” of product life cycles, which is more geared towards mass market commodity goods. For an introduction to the product life cycle literature in economics, see James M. Utterback and William J. Abernathy, "A dynamic model of process and product innovation", *Omega* 3, no. 6 (1975).

[56] On the role of standardisation in history, see Andrew L. Russell, "Standardizing in History: A Review Essay with an Eye to the Future", in *The Standards Edge: Future Generations*, ed. Sherrie Bolin (Ann Arbor: Sheridan Press, 2005); ———, ""Industrial Legislatures": Consensus Standardization in the Second and Third Industrial Revolutions" (The Johns Hopkins University, 2007); Amy Slaton and Janet Abbate, "The Hidden Lives of Standards: Technical Prescriptions and the Transformation of Work in America", in *Technologies of power: essays in honor of Thomas Parke Hughes and Agatha Chipley Hughes*, ed. Michael Thad Allen and Gebrielle Hecht (Cambridge, Mass.: MIT Press, 2001).

makes sense to also understand programming language technology as tools and, in essence, capital goods.[57]

Following this, it should be possible to reconcile Chill's life cycle with the life cycle pattern typical to capital goods. However, programming languages are also a very peculiar type of capital good: they are intangible, abstract and at first highly malleable. When diffused, however, changes to the language are an intricate matter, as they could easily break existing implementations and code. As such, innovation and obsolesce of this technology appears similar to that of complex products and services.[58] Andrew Davies has suggested that complex systems evolve through two phases of innovation. First, the development of new systems architecture comes prior to commercialisation of the product and another where the rate of components and systemic innovation increases and new products and components are introduced, without fundamentally altering the established architecture. Following this, it is argued that innovation happens through a long period of time. In many ways, programming languages are similar to such architectures or platforms that a technological system forms around or is built on top of.

Throughout the thesis I refer to the combined process of institutionalisation and technological change as bound up in processes of *technical diplomacy*. Here, both the figurative and the direct meaning of the term diplomacy are applied to the task. The concept refers to the extensive negotiations that went on among technological practitioners and scientists, discussed at an international level within the ITU, a process very much similar to that of diplomatic negotiations of international relations: bargaining, standoffs, coercion and ratification of agreements were all part of the process. I also apply the term diplomacy to the general process of directing and shaping the future of programming languages in the telecommunication industry, here including strategic positioning among telecommunication organisations that went on outside the ITU arena and the small-scale negotiations on technicalities that happened at technical conferences and symposia around the world. As such, the concept includes diplomacy at large and diplomacy of the daily routine, beyond the phases of

---

[57] On the role of software as capital, see Howard Baetjer, *Software as capital : an economic perspective on software engineering* (Los Alamitos, Calif.: IEEE Computer Society, 1998).

[58] On the historical importance of capital goods, see Nathan Rosenberg, "Technological Change in the Machine Tool Industry, 1840 - 1910", *The Journal of Economic History* 23, no. 4 (1963). On complex products and services, see Andrew Davies, "The life cycle of a complex product system", *International Journal of Innovation Management* 1, no. 3 (1998); Mike Hobday, Howard Rush, and Joe Tidd, "Innovation in complex products and system", *Research Policy* 29, no. 7-8 (2000).

emergence. As such, the diplomacy related to whether the programming languages should be used is understood to be as important as the initial positioning.

The processes of technical diplomacy were shaped by the international telecommunication regime, which in turn influenced norms held among the participants in communities of technological practitioners and by the strategies and actions of telecommunication organisations such as telecommunication administrations, equipment manufacturers and research organisations. In the following, I will define each in turn.

Traditionally, the ITU has been described as the main key in an *international telecommunication regime,* a system that in retrospect has been named the "ancien regime" by some scholars.[59] This regime was protecting a system of national telecommunication monopolies and controlled by the telecommunication administrations.[60] Typical to writers analysing the ITU as part of such regimes is that they allow the individual participants little room for manoeuvre and little room for technical norms that would transgress the boundaries of the interest. This thesis questions this approach, as it assigns at least some agency to communities of technological practitioners, without making these communities independent of the established political and economic structure of the international telecommunication regime. Furthermore, as the organisation of telecommunications to some extent varied beyond the typical monopolistic regime described by Cowhey, in particular in the Nordic countries, there are further reasons to be cautious about accepting this description in full.[61] Still, the structural description of Cowhey is a valid one, just as the term "ancien regime" for telecommunications is an historical fact. What it questions in this thesis is the mechanisms assigned to this "ancien regime".

A *community of technological practitioners* is understood as a structure of both social and epistemological character, where its body of knowledge and its social dimensions are intertwined. Such communities form around a communally defined problem, which is gradually redefined by the practitioners who are attracted to it. They are often quite small, although varying over time, and non-exclusive. As such, their members might be active in several different communities. The non-exclusivity, problem-

[59] Drake, "The Rise and Decline of the International Telecommunications Regime".
[60] Cowhey, "The international telecommunications regime: the political roots of regimes for high technology".
[61] A pertinent reminder of the variety of organisational principles in the telecommunication industry is Andrew Davies, *Telecommunications and Politics: The Decentralised Alternative* (London: Pinter Publishers, 1994).

orientation and small size and informal membership process is what separates a community from a profession.[62]

The economist Dominique Foray has underscored the importance of the community as an organisational system allowing the exploitation of the properties of knowledge in his book *The Economics of Knowledge*.[63] According to Foray, knowledge-based communities, which are similar to my conceptualisation of communities of technological practitioners, are networks of individuals striving, first and foremost, to produce and circulate new knowledge, and working for different, even rival organisations. He has described their workings as the "machineries of knowing".[64] Foray claims that the relevance of these communities is increasing. However, it is obvious that such communities were also very much in effect in radical technological transformations, such as the transition towards computer-controlled switching systems in telecommunications. The historian of technology Edward W. Constant argued in the early 1980s about the importance of specialised, differentiated, well-defined communities of technological practitioners. To Constant, focusing on a technological community helped him explain and account for the turbojet revolution in aviation.[65] While Constant was inspired by sociological studies of the "invisible colleges" in science, he drew a sharp line between how community structures worked in relation to science and to technology. In my review of the interaction and intersections of various software communities, these boundaries seem less clear. The communities of technological practitioners do not exclude researchers, designers or users. The term "technological practitioners" has to be understood rather broadly.

Over time, these communities are held together by *norms*, understood as a set of common understandings or values that allow members of the communities of technological practitioners to choose among a set of options. These norms addressed what was held as important attributes of

[62] A similar application of the community term is found in Ann Johnson's application of the "knowledge community" framework in her study of anti-lock braking systems, where the basic argument is that communities are the basic locus of knowledge production in design engineering and much science. See Ann Johnson, *Hitting the brakes : engineering design and the production of knowledge* (Durham: Duke University Press, 2009).
[63] Foray, *Economics of knowledge*.
[64] Ibid., 183.
[65] Constant, *The origins of the turbojet revolution*.

programmers, be it a rigour, systemic thinking, creativity or speed.[66] In this particular setting, I refer to these norms as *development virtues*, and when practiced, the communities would inevitably refer to these practices as good programming. These norms created dispositions towards certain technologies or tools, and were able to attract or direct processes of technological change. Still, the term development virtue is solely an analytical concept that I use for reasons of explanatory power and clarity. It is not a concept that can be found explicitly in the vicinity of software developers, either in the historical period I investigate or today. However, it is not without precedents: just as objectivity has been deemed the most important *epistemic virtue* in western science, that is to say that objectivity has been understood as the most important moral attribute of scientists, several ways of programming have been elevated as virtuous ways of programming. In particular, programmers driven by ideals such as elegancy, minimalism and formalism have been held in high esteem, although the elevation has often been contested and is a source of conflict.[67] Development within different "communities of computing" differed, just as software development for finance institutions and military operations varied in strikingly different manners.[68] As these differences became obvious, new development virtues were established through community processes and directed communities and participants in their continued struggle with changing their ways of developing software. When moving software development into telecommunications, norms would have to be adapted to this new application domain, which in turn made room for heated debates. Typical of those opposing development virtues were some who wished for software development as a formal undertaking, or at least one that was rigorous and systematic, a view that often clashed with those inclined to a

[66] Here, I appropriate parts of the terminology used by the historian of science Peter Galison when studying what he calls epistemic virtues. See Lorraine Daston and Peter Galison, *Objectivity* (New York: Zone Books, 2007). Furthermore, I include the core concept of agendas found in work by Michael S. Mahoney, a historian of science that has written extensively on the history of software. See Mahoney, "Software: The Self-Programming Machine"; Michael S. Mahoney, "Software as Science - Science as Software", in *History of Computing: Software Issues*, ed. Reinhard Keil-Salwik Ulf Hashagen, Arthur L. Norberg (Berlin: Springer Verlag, 2002); ———, "Finding a history for software engineering", *Annals of the History of Computing* 26(2004); ———, "The histories of computing(s)", *Interdisciplinary Science Reviews* 30, no. 2 (2005).

[67] Such discussions are everywhere in contemporary literature, both from trade and from the sciences. For an illustrative overview, see Ensmenger, *The computer boys take over : computers, programmers, and the politics of technical expertise*.

[68] On the "communities of computing," see Mahoney, "The histories of computing(s)".

more artisan approach to software development. Such sets of norms could be shared across professional identities, be it computer scientists, professional programmers or software engineers.[69] In the following, I treat these various groups as part of different communities of technological practitioners.

Processes where technical features were decided on through diplomacy between community members were not the only ones that shaped Chill, as technological development is seldom enclosed to such a technical realm alone. It was just as much influenced by the organisational linkages of the participants and a number of factors more external to the Chill project. This includes the *strategies and actions of telecommunication administrations and manufacturers* in relation to common programming languages in general as well as to the Chill project more specifically. In particular, the strategies on the future division of programming labour, among administrations, manufacturers and independent firms, would influence language design. Throughout this thesis, I try to discern between how such strategising was acted upon by the very same participants who held strongly shared norms, and by actions carried out above their level of authority.

The main period under investigation, from the early 1970s to around 1990, was one where the organisational principles of telecommunications were also about to undergo radical changes, as the dominant pattern of a strongly regulated industry was put under increasing pressure. However, this did not imply a unification of strategies among telecommunication administrations or manufacturers. On the contrary, the general shift towards "deverticalisation" was not one adhered to by all organisations.[70] By focusing on a particular piece of the game, this thesis highlights the unruly strategic landscape that led to the dissolution of the "ancien regime" of telecommunications.[71]

In this thesis, I analyse how the relative strength of the two factors, the strategising of the telecommunication organisations and the norms of the communities of technological practitioners, oscillated throughout the periods of emergence, use and eventually demise – in short the full life cycle of the specific technology. Furthermore, I analyse how the unruly relationship between the dissolving international telecommunication regime, the communities of technological practitioners and the telecommunication organisations changed over time, and shaped Chill.

---

[69] Ensmenger, *The computer boys take over : computers, programmers, and the politics of technical expertise*.

[70] Langlois, "The vanishing hand: the changing dynamics of industrial capitalism".

[71] Drake, "The Rise and Decline of the International Telecommunications Regime"; Cowhey, "The international telecommunications regime: the political roots of regimes for high technology".

The theoretical apparatus accounted for above is related to three streams of research that are found within various incarnations of economic, sociological and historical theories of the *direction* of technical change, broadly defined. This involves theories that account for technological design as well as use, either by pointing out how the two phenomena differ or how they might be interrelated. I call these broad traditions "contingentism", "institutionalism" and "rationalism." By contingentism, I lump together theories that explain technical change by pointing to local processes that easily could have ended up differently, or different processes that could have ended up instigating the same result.[72] This view is shared in theories such as the social construction of technology and actor-network theory, both mainstays within so-called science and technology studies (STS).[73] By institutionalism, I expand the theoretical category of new institutional theory to comprise theories that explain technical change by trial and error processes and technological paradigms, including evolutionary economics and neo-Schumpeterian theories of innovation.[74] By rationalism, I group together theories that apply economic or technical rationality as a determining factor when explaining which technologies are being developed among a set of feasible changes, and typically conflate the question of diffusion to the same rational decision process.[75]

The move towards high-level programming languages for telecommunications, in general, can be explained by a rationalistic framework, as it successfully accounts for how a particular price structure induced innovations to follow route. Two traditions of this way of reasoning are typical. The first, following the lead of John Hicks, is a stream of research within what is generally known as the theory of induced innovation that dominated the 1960s and 1970s.[76] Secondly, and more recently, research associated with the so-called endogenous growth models has tried to seek

---

[72] Fittingly, the term contingency has been defined in a number of different ways in different research traditions. Most typically, it involves either the modal logic understanding of "neither necessary nor impossible", or the more casual interpretations of the word meaning either dependency or chance. In general, all the conceptualisations share a focus on indeterminacy and unpredictability.

[73] One early example of this is found in Wiebe E. Bijker and John Law, *Shaping technology/building society : studies in sociotechnical change*, Inside technology (Cambridge, Mass.: MIT Press, 1992).

[74] For an overview of the literature, see Fagerberg, "Innovation: A guide to the literature".

[75] See for example Vernon W. Ruttan, *Technology, growth, and development : an induced innovation perspective* (New York: Oxford University Press, 2001).

[76] For a short presentation and comparison of the institutional theories laid out above, see ———, "Induced Innovation, Evolutionary Theory and Path Dependece: Sources of Techncial Change", *The Economic Journal* 107, no. 444 (1997).

out how research and development can influence growth across countries – and as a part of this strived towards modelling technological change as a consequence of intentional actions taken by people who respond to market incentives.[77] The most explicit effort to explain the direction of technical change is associated with the Hicksian induced innovation theory, where the core idea was that "a change in the relative prices of factors of production is itself a spur to innovation and to invention of a particular kind – directed at economising the use of a factor which has become relatively expensive".[78] Following this, innovation was understood as a rational, goal-seeking activity, helped forward by an exogenously created knowledge frontier. The major limitation of this model was pretty obvious, also to its main proponents: the internal mechanism, such as the learning, search, research and development, remained unexplained – exogenous to the economic system. These mechanisms have recently been rectified, either through the integration of evolutionary theories, or through endogenous growth theories.[79] In the latter incarnations, the price effect that creates incentives for innovations in the familiar Hicksian way is supplemented by other factors when explaining technological direction, such as the market size effect, "which encourages the development of technologies that have a larger market".[80] Following this, the diffusion of innovations follows almost in an automatic fashion, whereas rational actors would apply the most effective technique available, and since innovations are understood as only happening if they save a given factor. In the case of Chill, such theorising can, to some extent, explain the initiative, as a common programming language could be a labour-saving technology. However, its sudden fall from grace would necessitate either the appearance of a more effective technology or radically changing factor prices. None of these explanations are very convincing. The explanation must take into account the changing strategies of established organisations, emergent communities of technological practitioners and the changing political-economic regime surrounding telecommunications.

The study of the technical diplomacy that shaped Chill throughout its life, be it the small-scale diplomatic bickering about programming language features or the large-scale diplomacy regarding the adherence to the Chill "treaty", is at the centre of this thesis. At first sight, this seems related to what is commonly held as important tenants of any variant of contingentism, where the negotiability of technologies has been a main concern. This is

[77] The new growth models stems, in large parts, from Paul M. Romer, "Endogenous Technological Change", *Journal of Political Economy* 98(1990).
[78] John Hicks, *The theory of wages* (London,: Macmillan, 1932), 124-25.
[79] See for example Daron Acemoglu, "Directed Technical Change", *The Review of Economic Studies* 69, no. 4 (2002).
[80] Ibid.: 793.

often found within work in what is now known as social studies of technology, and often also related to sociological studies of science.[81] Here, technology development is perceived as the outcome of interactions and negotiations between various social groups, but ordinarily eschews economic superiority and technical efficiency as criteria for change and choice. Innovation, the shaping of technologies and their use are understood as an outcome of controversies stemming from interests, strategies and knowledge among actors. [82] It is theorised that within these controversies, a technological framework emerges and is gradually shared among relevant social groups and then directs the process towards closure. This process is understood as contingent, which basically implies that the technology would have ended up rather different given other constellations of actors, while the shared set of technological frameworks is understood as fairly rigid and difficult to overcome. The latter is something that is built alongside the technological project and hence not predetermined.

However, I am not convinced that it is really necessary to eschew economic and technical rationality to favour contingent negotiations. I find that the explanatory power can be strengthened if one reconciles parts of the literature focused on the contingent aspects of technology with literature oriented towards the institutional aspects of technological change.[83] Within the stream I labelled institutionalism, the direction design and use of new technology stems from norms, technological trajectories, paradigms, and routines that can all be understood as institutions influencing decision-making. [84] All these concepts have been applied as explanations of the direction of technical change: In the evolutionary tradition following Nelson and Winter, routines and technological trajectories push evolution in given directions.[85] In another variant, Dosi has extended the Kuhnian notion of

---

[81] For an overview, see the various contributions in Bijker and Law, *Shaping technology/building society : studies in sociotechnical change*.

[82] Wiebe E. Bijker, *Of bicycles, bakelites, and bulbs : toward a theory of sociotechnical change*, Inside technology (Cambridge, Mass.: MIT Press, 1995).

[83] An attempt to reconcile the sociology of scientific knowledge, an important stream of research within the field of science and technology studies, is put forward in D. Wade Hands, "The Sociology of Scientific Knowledge: Some Thoughts on the Possibilities", in *New Directions in Economic Methodology*, ed. Roger Backhouse (London: Routledge, 1994).

[84] Here institutions imply the humanly devised rules of society, to paraphrase the economist and historian Douglass North, a main character in traditional institutional thinking and theorising. Douglass C. North, *Institutions, institutional change and economic performance*, The Political economy of institutions and decisions (Cambridge: Cambridge University Press, 1990).

[85] Richard R. Nelson and Sidney G. Winter, *An evolutionary theory of economic change* (Cambridge, Mass.: Belknap Press of Harvard University Press, 1982).

paradigms to encompass the field of technology, where the search for new products or processes is formed by an "outlook" that directs the efforts of technologists and engineers. To Dosi, "a technological paradigm embodies strong prescriptions on the directions of technical change to pursue and those to neglect".[86] A similar Kuhnian outlook is found in Edward W. Constant's historical research, which I already quoted as an influence when stressing the importance of understanding the community level of technological change. In his research, knowledge and norms held by communities of technological practitioners are the level where technical change emerges and its fate is decided upon.

Constant explicitly applied the concept of normal technology as an analogy to Kuhn's normal science, where the improvement of the accepted tradition or its application under new or more stringent conditions is akin to the "puzzle solving" of Kuhn. It is functional failures and anomalies in periods of normal technology that direct the emergence of novelty, and the model firmly places the directional element within the stable practice. In many ways, Constant shares this with the prominent historian of technology Thomas Hughes, whose outlook on the dynamic element in the evolution of large technical systems is innovation caused by reverse salients – understood as bottlenecks or weak spots observable in technical arrangements.[87]

When I emphasise the importance of norms at the level of the communities of technological practitioners, I also draw on another branch of the institutionalist literature. Within so-called neo-institutionalism, studies of how actors or organisations follow norms to gain legitimacy have been an important strand.[88] Parts of this literature have tried to explicate how collectively valued purposes can influence normative or moral legitimacy. In some ways, this can be reconciled with the Kuhnian outlook of Constant that I discussed above, as it adds a mechanism for understanding individual and organisational behaviour within paradigms, as the value of gaining or retaining legitimacy are understood as what directs actions.

Although the similarities between my approach to the design and diffusion of Chill is similar to many institutionalist theories, it does not share

---

[86] Giovanni Dosi, "Technological paradigms and technological trajectories", *Research Policy* 11(1982): 158.

[87] Thomas Parke Hughes, *Networks of power : electrification in Western society, 1880-1930* (Baltimore: Johns Hopkins University Press, 1983).

[88] The new institutionalism's interest in legitimacy started with John W. Meyer and Brian Rowan, "Institutionalized organizations: Formal structure as myth and ceremony", *American Journal of Sociology* 83, no. 2 (1977). Following this, a huge literature has emerged and expanded in many directions. See for example David L. Deephouse and Mark Suchman, "Legitimacy in Organizational Institutionalism", in *The Sage Handbook of Organizational Institutionalism*, ed. Royston Greenwood, et al. (London: Sage Publications, 2008).

this approach towards periodisation: macro-based periodisation is a demarcation strategy common to much historical thinking, and similarly something held by many institutionally oriented scholars. Typically, some periods are thought to be characterised by a dominant technology such as "the age of steam" or "the internet era". Others periods are characterised by radical change and a particular technical breakthrough, such as the industrial revolution or the rapid electrification of the late 1800s. Such an understanding of technological periods is bound up in an idea that basic innovations cluster in time, which again has resulted in the popular idea that the appearance and diffusion of innovations and technical change as an uneven process, sometimes gradual and sometimes explosive.[89] This model is shared by many writers within the Schumpeterian tradition of studies of innovation, and the more institutional approaches in general.[90] It was also popular in particular in the so-called "long-wave" theories that were somewhat in vogue from the late 1970s and into the 1980s.[91] Typically, such period frameworks are based on the stability of "normal technology" versus periods of revolutionary innovations.[92] At the micro-level, similar two-pronged models are just as common, often putting up sharp dichotomies between phases of innovation and phases of diffusion within technological life cycles.[93]

In this thesis, I argue that this two-pronged pattern is unsatisfactory, at the micro- as well as at the macro-level. I put forward the proposition of a three-pronged model, where one can argue that decision-making about new and novel technologies can be shaped by various "period specificities", either in forms of stable technological paradigms, to use Kuhn's term describing normal science, revolutionary upheaval or Turnerian liminality.[94] To the practitioners involved in processes of technical change, some periods are not mainly about "puzzle solving" or decision-making under complete uncertainty. Somewhere in between lie periods of transition between

[89] This is the very heart of the writings of Joseph Schumpeter. On the clustering of basic innovations in time, see Schumpeter, *Business cycles; a theoretical, historical, and statistical analysis of the capitalist process*, 75.
[90] On a direct appropriation of the Kuhnian model of change in sciences, see Dosi, "Technological paradigms and technological trajectories".
[91] For a recent example and a summary, see Christopher Freeman and Francisco Louçã, *As time goes by : from the industrial revolutions to the information revolution* (Oxford ; New York: Oxford University Press, 2001).
[92] On normal engineering, see Constant, *The origins of the turbojet revolution*.
[93] See, for example, Steven Klepper, "Entry, Exit, Growth, and Innovation over the Product Life Cycle", *The American Economic Review* 86, no. 3 (1996).
[94] On scientific paradigms, see Thomas S. Kuhn, *The structure of scientific revolutions*, 2nd ed. (Chicago,: University of Chicago Press, 1970).

somewhat stable states.[95] Such periods are perhaps best compared to periods where one navigates towards a somewhat known goal, but where the waters are uncharted. At the micro-level, such uncharted waters are typically found when new technology is moved from a phase when one goal dominates, to another phase, where more and more new targets emerge. Such phases are found in the period of time when a new process is moved from the testing laboratory to the production line, or when a programming language for telecommunication switching undergoes extensive trial implementations, after a period of innovative design. Such liminality is a common feature in processes of technical change and innovation, perhaps more so than in other types of social activities. Periods of liminality mean complex and novel decision-making for technological practitioners, not necessarily fitting the grand scheme of things made up by historians and social scientists. It is this category of periods that is the focus of this thesis. Here, periods and technologies are understood less in terms like monarchic successions, but more as "new stars twinkling into existence, not replacing old ones but changing the geography of the heavens", to paraphrase an imaginative illustration of the way modes of scientific reasoning have changed over time.[96]

Chill was initiated in what might perhaps best be described as such a liminal phase at the macro level, at the outset of the knowledge-based economy, where the meaning of programming as an activity to both manufacturers and administrations was unknown. It was really "betwixt and between" positions assigned by prior knowledge, technical solutions and organisational patterns.

At the micro level, it is possible to recognise similar "betweenness," typical in periods moving on from inventive design to implementing use and from rapid growth to stagnating diffusion. In the following, I apply a three-pronged micro-periodisation when comparing how Chill was shaped through periods of invention, innovation and diffusion, but pay particular attention to the periods in between these well-defined phases.

Many of the discussions over the last three decades on social scholarship on the direction of technological change have been about the apparent gulfs between the three main streams pointed out above, the

[95] Turner, *The forest of symbols; aspects of Ndembu ritual*.
[96] Daston and Galison, *Objectivity*, 18.

"contingentism", the "rationalism" and the "institutionalism".[97] Changes within social studies of technology during the last three decades have almost altogether rejected the notion of a direction of technical change (and even more so, trajectories) as a plausible term when considering technological change, mainly because of its implication of technological determinism. On the other hand, the traditional concept of directed technical change within economics has been revived, in particular as economics has turned towards the problems of technological solutions to climate change.[98] In the following, I am equally interested in the possibilities inherent in these established theoretical streams as in their limitations.

## Methods

The following presentation of the methodology is organised around three general principles, concerned with the design of the investigation, its level of analysis and the sources used in the analysis. Although they are formulated in rather general terms, they are intended for the applicability to the project at hand rather than generality.

Sticking point number one is to reconstruct the historical sequence of events by following the technology through primary sources. Principle two is to compare across time and space. The third principle is to validate the historical reconstruction and the comparison through secondary sources and contemporary data. I will deal briefly with each principle in turn.

The methodological principle number one is concerned with the level of analysis and the sources used in the analysis, as it states that I should reconstruct the historical sequence of events by following the technology through primary sources. By this, I imply that the study follows the technology rather then a set of predetermined set of actors. I have tried to reconstruct the constantly changing communities and organisations that exerted influence over Chill over time. This implies that the theoretical model sketched out above is primarily sought out at a level of individual

---

[97] On the gulf between evolutionary and mainstream economics regarding technological change, see Fulvio Castellacci, "Evolutionary and new growth theories: Are they converging?", *Journal of Economic Survey* 21, no. 3 (2007). On the differences between social construction of technology and evolutionary approaches, see Odd Einar Olsen and Ole Andreas Engen, "Technological change as a trade off between social construction and technological paradigms", *Technology in Society* 29(2007).

[98] Most explicitly rejected in the introduction in Bijker and Law, *Shaping technology/building society : studies in sociotechnical change*. On the interest in directed technical change and its relation to climate change, see the various contributions in Arnulf Grübler, Nebojša Nakicenovic, and William D. Nordhaus, *Technological change and the environment* (Washington, DC: Resources for the Future ; International Institute for Applied Systems Analysis, 2002).

actors rather than one where the strategies of firms and the institutional framework can be appreciated from a vantage point high above the heads of the participants. On the contrary, the way I have chosen to follow the technology allows an understanding of organisational strategies and institutional regimes as they appear at the level of individual participants. This makes this history almost a history from below, regardless of its insistence on not following a prescribed set of actors.

This principle renders the investigation of little use if I were to explain differences in how groups of actors behave, or the variance in the level of competitiveness or performance of some firms or nations. However, I find the principle all the more fitting when concerned with explaining the direction of technical change, although such a strategy might have some unfortunate consequences when dealing with something as esoteric as a programming language. The technical vocabulary might seem alien even to technically gifted readers, and impenetrable to those not well versed in the lingo of programming. I have tried to keep the discussion as non-technical as possible. However, the aim of opening the black box of something as heterogeneous and specialised as programming language design prevents me from bypassing technological arguments and conflicts as something wholly esoteric.[99]

The historical reconstruction is based on an extensive use of primary archival sources. They are mainly of two kinds: One category of sources embodies the technological project, such as working documents and plans, and is of a descriptive character. These sources originated in the technical project itself. Another type of documentation says something about its originator and author, like travel reports and written correspondence between developers and users, which can in some cases reveal intentions, hopes and values, information that is important to the historical accounts. Such sources embed part of the social fabric that the participants were woven into. Together, the breadth and number of sources makes a detailed reconstruction possible. I discuss the limitations and possibilities of the available sources more directly in a specific section below.

It is also worth noting that although I rely primarily on archival sources, I also activate a slightly broader methodological apparatus throughout the thesis. Interviews have been used to some extent as part of the reconstruction, which I will deal with more extensively below. When analysing relationships between participants in the ITU design and implementation projects, I utilise so-called social network analysis (SNA) to

[99] On the black box metaphor in relation to technology, see Rosenberg, *Inside the black box : technology and economics*.

discern patterns of cooperation and alliances.[100] Here, I focus on the relationships formed through mutual participation in the official meetings and the contributions made by team members, or their parent organisation, to different meetings. This makes it possible to distinguish between importance (or centrality, to use the lingo of SNA) won through participation and the willingness to exert influence through contributions to the various meetings. The former is understood as a position in a network made up of ties between the participants and the latter is understood as an indication of a willingness to exert influence over the decisions made at each individual meeting. Throughout chapters three and four, I measure participation networks and the willingness to influence decisions through such methods. More thorough discussions on the methodological possibilities and limitations of such network analysis and its measures are carried forward throughout the relevant chapters.

The social network analysis is coupled with a close to complete analysis of various written contributions to the programming language, which makes it possible to compare how linkages between participants were reflected in contributions, beliefs and behaviour in the committee work, which adds to the gains from the social network analysis. It is worth noting that the social network analysis is only appropriate when analysing and comparing the shaping of the programming language that was going on within the boundaries of the ITU, and is not applicable when analysing the later stages of Chill's life cycle. The analysis of its wider use and adaptation has to rely on other methods, like comparing patterns of use across various organisations.

The second methodological principle is to compare through time and space. The purpose is to come up with a descriptive explanation of the problem at hand, an explanation that is inherently historical and dynamic in the sense that I amplify the contextual and the period specifics of the events analysed. The time-oriented comparison is based on a periodisation of the development of the Chill project, and tries to compare the directional push towards a specialised, yet high-level programming language, along the two sources of influence discussed above.

The "life cycle" of the Chill programming language lends itself to quite distinct periodisation, as the institutional base of Chill, the ITU, was strictly organised around a principle of pre-planned "study periods" with an interval of four years. The organising principle of the time-based comparison

[100] SNA is generally considered a well-established method within the social sciences that is particularly useful when considering community structures and analysing relational data. For an introduction, see Linton C. Freeman, *The development of social network analysis: a study in the sociology of science* (Empirical Press, 2004).

is, as such, internal to the study object, rather than based on a broad and externally originated period or phase.

The period-oriented comparison is also, more inherently, a comparison across space. The level of analysis, including communities of technological practitioners, the international organisation ITU and nationally bound administrations and manufacturers, allow me to seek out common concerns and actions across boundaries typical to national-oriented analysis of innovation.[101] As such, my approach allows for the international and transnational aspects of technology development as well as use to come to the foreground. However, it also makes it possible to compare how actors coming from widely different organisational and national backgrounds acted at that international level, and thus being able to answer whether the convergence upon some technical features was a result of tough negotiations or common agendas.

The third methodological sticking point is basically a safety valve: I have tried to validate my findings and my interpretations of primary sources through secondary sources. This includes an extensive use of contemporaneous technical and scientific literature and retrospective interviews with some of the key actors in the Chill project. In this regard, I have relied on interviews with important actors in the Chill project to strengthen my analysis. This includes some important Norwegian, Swedish, Dutch, Japanese and American participants, but includes nowhere near what would be demanded of a representative population of interviewees.[102] As such, it is important to stress that the way I have utilised these interviews has mainly been to validate information found in archival sources, or to point out relationships that were not evident from the primary sources in the first place. As such, there are parts of the analysis that rely more heavily on interviews than others. In these cases, I have tried to support the evidence with other interviews and secondary sources as much as possible.

These interviews have been done orally and in an unstructured way, except in some cases, where conversations by electronic mail have been used due to distance and time. Some of the interviews and conversations have, following the validation criteria, taken place late in the research process. In some parts of the text, these interviews have provided me with unique knowledge and information, which in turn has informed my understanding in

[101] On the deficiencies of a national-oriented approach to innovation and use of technology, I find David Edgerton's arguments convincing. See David Edgerton, *The shock of the old : technology and global history since 1900* (Oxford ; New York: Oxford University Press, 2007), 103-37.

[102] For more details, see list of interviews in the appendix. In the running text, references to interviews have been given the following form: Name of interviewee, interviewed by author, date of interview, and place of interview.

a considerable way, although they appear infrequently in the referring footnotes.

When I have used direct quotes from interviewees in the text, the quotations have been approved by the interviewees. The evidence used for most parts of this thesis has been in written form and from the time they cause concern. The unwillingness to rely more on interviews and oral history methods is based on the considerable problems with the reliability of such interviews, in particular due to the sketchy reliability of memory and the problem of retrospective interpretation. This is, however, not due to the specific informants available to this specific project, but a general scepticism towards the use of interviews as evidence on my behalf. The interviews conducted for this thesis have also played a significant role in identifying written sources and provided access to private collections of papers and communications. As such, the interplay between interviews and written sources has to some extent gone beyond the validation-oriented principle stressed above.

Summing up, the methodology applied aims at reconstructing the Chill project through the use of mainly archival sources; it compares the directional push to the technology through its life cycle, and validates the specifics and general implications through secondary and contemporary data. Let me briefly consider the strengths and weaknesses of the archival sources in a separate section below.

## Sources

As discussed above, the historical reconstruction is based on an extensive use of primary sources. These are sources that embody the technological project, such as working documents and plans, and are of a descriptive character, or documentation that says something about its originator and author, like travel reports and written correspondence between developers and users that can in some cases reveal intentions, hopes and values of their originator. The two categories of source material have been found in a number of different archives and private collections. First, the official archive of the ITU has made available the official documentation from the Chill project, meaning the working documents from the study group from around 1973.[103] The ITU material in this period consists of approximately 22 metres of shelving, and includes material from all working groups active during the study periods, but not the final publications. However, only a small fraction of this material is relevant for my study. They lack detail, because of ITU's policy of archiving only documents of the official working group and the documents submitted to them within a particular timeframe.

[103] International Telecommunication Union Archive, Geneva, Switzerland (hereafter cited as ITUA).

As such, what the ITU termed delayed contributions is not held by the archive. Reports on meetings within the working group structure are also found within ITU's journal, the *Telecommunication Journal*, which has been consulted for all periods analysed. Official documents from ITU's plenary assemblies have been examined for the relevant periods and the final publications of the Chill recommendation.[104] Furthermore, I have also used some periodicals relevant to the subject matter. First, the ITU publication, the *Telecommunication Journal*, which was published monthly in Geneva, has been looked through for the years 1975 to 1985.[105]

Secondly, I have looked into material in the archival holdings of the Norwegian Telecommunication Administration (NTA) held at the Norwegian National Archive.[106] This includes material held by the administrative department of the administration as well as its research establishment, the Norwegian Telecom Research (NTR).[107] In addition, I have studied records from the technical department as well as copybooks and journals from the central administration. The NTR coordinated the common Nordic efforts in the ITU on Chill, but the archival holdings are sporadic and incomplete. Some valuable insights into the administration's view on programming and its importance for future telecommunication networks are nevertheless possible to find here. This archive also held invaluable material for the later years of my study, in particular regarding efforts to commercialise Chill-related products.

Thirdly, I have enjoyed full access to the private collection of Kristen Rekdal, who was the main Norwegian researcher active in the Chill development work from 1975 and headed the international standardisation work from 1980 to 1984. He started the firm Urd and Kvatro, which commercialised some Chill products from 1983.[108] This archive amounts to roughly three metres of shelving, related to the CCITT project, as well as a

---

[104] Revisions to the original recommendation were issued in 1984, 1988, 1992, 1996 and 1999.

[105] The *Telecommunication Journal* was previously called the *Journal Télégraphique*, which had been issued from 1869. The *Telecommunication Journal* was launched in 1933.

[106] Norwegian Telecommunication Administration (Teledirektoratet in Norwegian), Norwegian National Archive (Riksarkivet in Norwegian), Oslo (hereafter cited as NTA).

[107] The latter is held in the following archive: Norwegian Telecom Research (Teledirektoaretet, Televerkets forskningsinstitut in Norwegian, hereby cited as NTR), Norwegian National Archives, Oslo.

[108] Private collection of Kristen Rekdal (hereby cited as KRC). The archive is about to be handed over to the archive of the Norwegian University of Science and Technology, Trondheim. I have named the boxes and binders in full, as they appear now, to ease finding the sources in any future reorganization of this collection.

supporting project done within Nordtel, an organisation for cooperation between telecommunications administrations of Norway, Denmark, Finland, Sweden and Iceland. Part of the archive also holds official material from the ITU - far more material than is available in the official archives of the ITU. The archive also holds several series of travel reports and commentaries about the progress of the work, material that has be harnessed to understand the ideals, beliefs and plans of its Norwegian originator.

Fourthly, I have looked briefly into the private collections of two Dutch participants in the Chill project, Remi Bourgonjon and Kees Smedema.[109] In the first case, this helped me complete the series of ITU-issued material, while in the latter case, internal documents of the Dutch firm Philips was made available. Both holdings were fragmentary and of a personal nature, and consequently they did not lend themselves to any systematic investigation. Information about decisions made within Philips and the organisational setting of the participants from Philips in the ITU project was obtained through these consultations. For more details about the archival sources utilised in this thesis, see the list of sources at the end of the thesis.

Additionally, this thesis makes use of a large number of journal papers, conference papers and technical reports to highlight how many organisations used and regarded Chill. In an industry that in this period remained fairly closed to outside observers, such sources have proven invaluable. In particular, I have looked into the complete proceedings from the Software Engineering for Telecommunication Switching Systems (SETSS) conferences, from its first incarnation in 1973 and up until 1992. Eight conferences with about 30 papers each have been indexed and analysed.[110] Furthermore, the proceedings of the larger industry meeting of the International Switching Symposium (ISS) have been analysed from 1972 to 1990.[111] Another important venue was the Chill conferences, five of which were held five times from 1981 to 1990. The papers given at these conferences has been analysed in detail and have revealed many details about the diffusion and use of Chill in almost all of the organisations that

[109] Both Remi Bourgonjon and Kees Smedema live in Heeze in the Netherlands.

[110] The Software Engineering for Telecommunication Switching Systems conferences were held in 1973, 1976, 1978, 1981, 1983, 1986, 1989 and 1992. They were organised by the Electronics Division of the Institution of Electrical Engineers (IEE), which became the Institution of Engineering and Technology in 2006. The various SETSS conferences were organised together with various other European institutes and associations around Europe and all the proceedings were published by the IEE.

[111] In this period, the ISS were held in 1972, 1974, 1976, 1979, 1981, 1984, 1987 and 1990 at various venues around the world. All the proceedings have been published.

used the programming language.[112] The specific articles that have been used are listed in the bibliography while a full article index of the Chill conferences, the proceedings of which were not widely circulated, has been included as an appendix.

One weak point in the sources that have been available to me is the lack of corporate archives from some of the large telecommunication manufacturers that were active in the Chill project. Such archives could have strengthened the understanding of the strategic manoeuvring of these firms and how that translated into actions within the Chill project. However, parts of the source material that have been available to me have gone some way to rectify the lack of corporate sources. First of all, by utilising the material submitted to the working groups within the ITU, it was possible to get information from a much larger set of manufacturing firms than a single corporate archive could offer. Secondly, due to the tangled nature of the telecommunication industry of the time, material from some manufacturers was available in the NTA archives. This does not, however, make up for the lack of corporate archives. In some special cases, in particular in the case of the multinational ITT and Philips, I have had to rely on interviews and conversations with people with a background in the Chill project, and the use of private collections of papers, notes and communications. For information about the Swedish firm L. M. Ericsson, I have also been able to make use of source material from the Nordic Chill compiler project, where Ericsson played a minor part. This has also been supplemented by source material made available through a large research project on Swedish IT history between 1950 and 1980, which has produced a number of transcribed witness seminars where former Ericsson employees have been extensively interviewed.[113] The public availability of these sources has, together with systematic study of the technical journal *Ericsson Review*, been able to further my knowledge about what went on in L. M. Ericsson.[114]

---

[112] The Chill conferences were held in 1981, 1983, 1984, 1986 and 1990. The proceedings very not officially published, but have been made available to me through the Kristen Rekdal collection, except for the proceedings of the first conference in 1981, which I have not been able to allocate.

[113] In particular, the following transcripts of witness seminars have been utilised: Per Lundin, "Tidlig programmering : Transkript av ett vittnesseminarium vid Tekniska museet i Stockholm den 16 mars 2006", (Stockholm: Filosofi och teknikhistoria, 2007); Mikael Nilsson, "Staten och kapitalet: Betydelsen av det dynamiska samspelet mellan offentligt och privat för det svenska telekomundret : Transkript av ett vittnesseminarium vid Tekniska museet i Stockholm den 18 mars 2008", (2008).

[114] The *Ericsson Review* was published four times a year and contained technical articles written in a fairly approachable manner. The journal was first published in 1923. I have looked at volumes 46 to 76.

The lack of corporate archival sources points out what can also be perceived as a methodological weakness, namely that the contextualisation of the Chill project in general and the approach to organisational strategies in particular rely, to a large extent, on secondary sources. When approaching projects involving as many organisations and participants as Chill, such problems are almost unavoidable. Constraints in time and space make a more detailed study of each and every organisation difficult to achieve. In this thesis, the main approach has been to use source material that originated between the participating organisations, close to the project that is. As such, what is studied is strategising as it appeared from below, which is the vantage point that this thesis has favoured. Still, I have tried to reconcile this with an extensive use of secondary sources, sources that are not only interviews, but also the large and extensive historical literature mentioned above.

## Outline

In this introduction, I have presented my subject and research problems, my methods and the theoretical orientation of my study. I have also briefly given a historiographical overview of relevant research. The rest of the thesis is organised as follows. The thesis is organised around three main parts, where each part is concerned with one of the three periods in the life cycle of the programming language Chill: its emergence, diffusion and demise.

The second chapter presents the broad technological and organisational background of the influence of programming and computing in the world of telecommunication administrations and manufacturers. Here, I also discuss differences between the two domains and put these into an historical perspective, in particular focusing on the years leading up to the Chill initiative, from around the mid-1960s. In particular, this chapter describes the state of affairs at the very first years of the 1970s, and outlines the invention of high-level and real-time programming languages. Furthermore, I seek out the first seeds of the communities of technological practitioners that shaped Chill, and analyse their institutional boundaries.

Chapters three and four are concerned with the period of emergence of high-level programming languages in telecommunications in general and the design of the Chill programming language in particular. This period is studied in detail, as decisions on programming language design is sought both in the internal dynamics of the international standardisation effort and in the external pressure from the participating organisations as well as external experts and knowledge communities. Chapter three is focused on programming language design, while chapter four focuses on how the language was implemented early on, and how this rubbed off on the language design.

Chapter five is an analysis of the early diffusion, use and implementation of the programming language Chill in large industrial firms. Chapter six focuses on the diffusion among administrations as well as how a

community of core language developers and more peripheral industrial users tried to influence the future of the programming language during the 1980s. Together, these two chapters study the phase where Chill matured into a language that was put into real use.

Chapter seven is concerned with the final diffusion phase and the ultimate demise of Chill. Here, I discuss a number of organisational alternatives for diffusion, in particular through small start-ups and the emergence of independent tool suppliers. I also discuss the way the impetus behind the language waned, and how it ultimately disappeared as a maintained ITU standard in 1999. Chapter eight presents my conclusions.

# 2. When switches became programs: telecommunication and computing, 1965 - 1974

In late 1974, delegates to the CCITT decided that a special purpose programming language for telecommunication switches should be created under the auspices of a committee appointed by the organisation. A common and internationally standardised programming language was thought to be an important way of unifying forces against the towering difficulties experienced when programming the very first generations of computer-controlled telecommunication switches.[1] No existing programming language fitted the bill drawn up by the members of the CCITT. Not technically, not economically, not politically. It was simply not possible to match the ideals and expectations that the delegates had of a special purpose programming language for telecommunication switching systems.

Programming language design was a field where the CCITT delegates and the employees of telecommunication administration held little experience or knowledge. The knowledge of the telecommunication manufacturing industry was also limited. [2] Programming a telecommunication switch was considerably different to programming a general computer, and little knowledge on how to do it existed even within the general computer industry.

The effort of standardising a programming language within the CCITT was just one of many ventures into the unknown waters of programming language design for telecommunication equipment. In many ways, such programming languages became “boundary objects”, spanning and mediating the borders of different technological and scientific communities and explicating different means of managing the complexity of programming telecommunication switches.[3]

---

[1] In the terminology of telecommunications, this first generation of switches was often referred to as Stored Program Controlled (SPC) telephone exchanges.

[2] The first computer-controlled switching system, the AT&T “Number One Electronic Switching System” (No. 1 ESS), is described in numerous sources. See for example the contemporary description in Bell System Technical Journal 43, no. 5 (1964). The Bell System Technical Journal is available in its entirety online, see http:// bstj.bell-labs.com/

[3] The concept of “boundary objects” was first used in Susan Leigh Star and James R. Griesemer, "Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39", *Social Studies of Science* 19, no. 3 (1989). It has recently been applied in the theoretical field of communities of practice. For an example, see Etienne Wenger, *Communities of practice : learning, meaning, and identity*, Learning in doing (Cambridge, U.K. ; New York, N.Y.: Cambridge University Press, 1998).

This chapter sketches out the background of the decision to create a CCITT-approved programming language for telecommunication switching. It charts the waters of designing and governing programming languages in general and outlines the challenges and solutions to programming switches sought during the first 10 years of computer-controlled switching, up until the CCITT decision in the mid-1970s.

Drawing on the framework established in chapter one, I stress the community as an organisational principle for understanding the development of programming languages. Consequently, I pay simultaneous attention to the social dimension of the community formation and how new knowledge was produced within it, the development virtues held by different communities, and how the borderlines between the two industrial sectors of computing and telecommunications became a zone where virtues clashed and new ones were established – virtues that again would influence the development of the CCITT-approved programming language.

## Programming languages in science and industry

In the early days of computing, software was not identified as a particular or isolated aspect of it. This changed in the late 1950s. As pointed out by the historian Paul Ceruzzi, computer programming, and consequently the whole software concept, was something contingent and emergent which came to the fore around 1959:

> The activity known as computer programming was not foreseen by the pioneers of computing. During the 1950s they and their customers slowly realized: first, that it existed; second, that it was important; and third, that it was worth the effort to build tools to help do it. These tools, combined with the applications programs, became collectively known as 'software,' a term that first came into use around 1959.[4]

This realisation was closely related to the emergence of computer use outside the laboratory, in industry and businesses. This had two implications: New applications of computing meant that programming became an activity that translated real-world problems into computable terms. It also meant that the programming activity of computers became an activity of the many. This caused the birth of a profession of programmers as well as an interest in building tools to help the programmers with programming software. In particular, this caused an interest in programming languages that could bridge real-world problems into computable expressions and constructions. These languages became the primary means of managing the complex

[4] Paul E. Ceruzzi, *A history of modern computing*, History of computing (Cambridge, Mass.: MIT Press, 1998), 108.

process of programming computer systems.

Initially, programming was a task requiring an understanding of obscure machine codes, a language few were to master. To ease the tedious and esoteric work of machine code, various types of notations were created, which was what we now call programming languages. That gave way to a more general notion of high-level computer languages, which implied that they were somewhat more readable to a human than to a machine, and that there existed a hierarchical system of programming languages, where the high-level programming languages were understood as something "above" the machine code and the intermediate level of so-called assembly languages. The high-level languages' expressions and constructions represented an abstraction of the computer, and were closer to the real-world problems than obscure machine codes. The instructions written by a programmer in a high-level language could be translated into machine code by the computer, by generating (or compiling, in computer lingo) machine code based on a careful analysis of what the programmer specified in such a high-level language.[5] This came, however, at a price: high-level language brought inevitable penalties in terms of size and performance of the compiled code and could run slowly on limited computing powers. The gains were increased programmer productivity, fewer bugs in the compiled code and better communication among programmers. Some degrees of machine independence of the code could be achieved, as solutions to specific computable problems could move more or less freely between specific computers.[6] For a casual illustration of the hierarchy of programming languages as of the 1960s, see below.

[5] To be precise, the code could either be compiled or interpreted. Compiled code means that the programming language expressions have been translated into "object code" that can run on a specific machine. Interpreted code is, on the other hand, executed on a step-by-step basis as each statement is translated into object code "on the fly" rather than in one batch. On the definitions, see Sammet, *Programming languages: history and fundamentals*, 12.

[6] On the early history of programming languages, see Ibid.

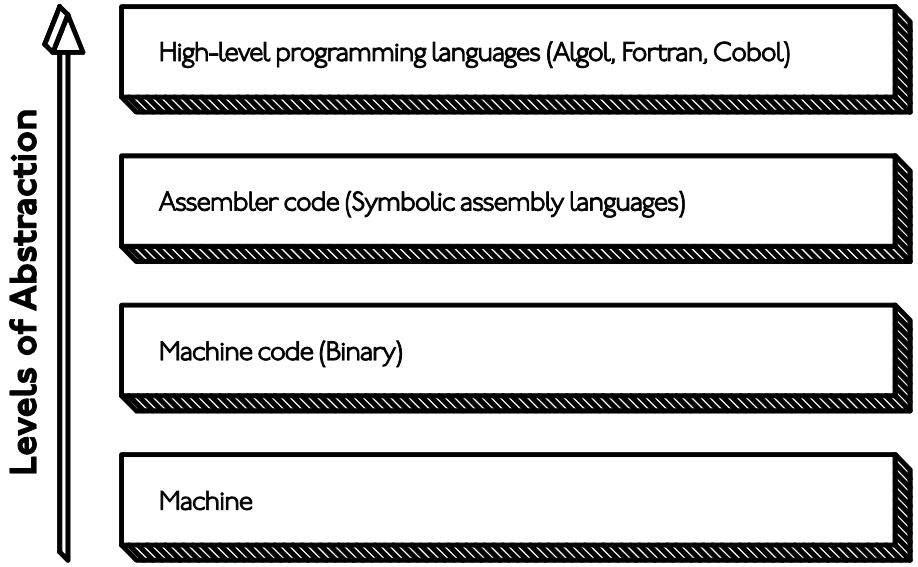


**Figure 2.1 Programming languages and the levels of abstractions**

The design of such high-level programming languages was a heterogeneous activity involving people from radically different backgrounds. Different agendas were followed when designing programming languages, agendas that were rooted in different software development virtues and ideals about how programming should be done. Some communities were interested in the efficiency of the code, held fast calculations as an ideal and the way programming should be done was considered secondary to the code. Others were mainly interested in applying mathematical algebra to the task, idealising programming as some sort of mathematical activity. As programming language design became an activity that attracted people with rather different backgrounds, the programming languages became boundary objects spun off and shared between communities of technological practitioners, scientists and other types of experts. Such a boundary object meant different things to different people. At one of the very first discussions of the programming language concept, a 1954 conference on programming organised by the US Navy, differences in the understanding of what a programming language should be were very apparent.[7] Most of the papers delivered at the conference were concerned with techniques for programming specific computers, while a few approached the task with the

[7] Mathematical Computing Advisory Panel United States Navy, *Symposium on automatic programming for digital computers, 13-14 May 1954*. (Washington,: U.S. Dept. of Commerce, Office of Technical Services, 1955). I was made aware of this conference and its illustrative powers in discussions with, and from article drafts by Gerard Alberts and David Nofre.

aim of liberating the programmer from the specificities of one machine, by applying more universal formal notation. In a paper by Saul Gorn, this was spelled out rather explicitly:

> As an alternative to the commercial capture of the computer and data processing field by one make of machine, or arbitrary ruling on machine specifications by government fiat, one now has the interesting possibility of a common, universal, external language arrived at by mutual agreement and persuasion.[8]

The solution, to Gorn and to a few other participants, was to root this universal language in the language of mathematics.[9] Not only that, Gorn invoked the tools of mathematics as a way of abstracting away the manufacturers' control over computer machinery. In the following years, those two issues, the application of mathematical logic and algebra to programming language design and the way of abstracting away the concrete machinery through universal language concepts, would form a central part of what basically became known as computer science in the USA and informatics in many European countries.[10] As the historian Michael Mahoney has argued, computer science formed upon an amalgam of a number of knowledge fields, such as numerical analysis, algebra, automata theory and computational complexity.[11] An overreaching issue for computer scientists was, regardless of the amalgamate background, to separate the issues of software from engineering and to strengthen the understanding of programs as mathematical expressions that could be proved right.[12] One way that academic computer scientists tried to enforce their particular view about what should be understood as good programming was through the design of programming languages. By injecting a particular logic and vocabulary into the most important capital good in programming, one could achieve the aims

---

[8] Saul Gorn, "Planning Universal Semi-Automatic Coding", in *Symposium on automatic programming for digital computers, 13-14 May 1954.* , ed. Mathematical Computing Advisory Panel United States Navy (Washington: U.S. Dept. of Commerce, Office of Technical Services, 1954), 75.

[9] J. Brown and J. Carr III, "Automatic Programming and its Development on the MIDAC", in *Symposium on automatic programming for digital computers, 13-14 May 1954.* , ed. Mathematical Computing Advisory Panel United States Navy (Washington: U.S. Dept. of Commerce, Office of Technical Services, 1954).

[10] On the formation of computer science and informatics, see Paul E. Ceruzzi, "Electronics Technology and Computer Science, 1940-1975: A Coevolution", *IEEE Annals of the History of Computing* 10, no. 4 (1989); Mahoney, "Software as Science - Science as Software"; Ensmenger, *The computer boys take over : computers, programmers, and the politics of technical expertise*, 111 – 36.

[11] Mahoney, "Software as Science - Science as Software".

[12] On the issue of provability in the history of computing, see MacKenzie, *Mechanizing proof : computing, risk, and trust*.

of virtuous programming. However, the formalistic development virtue that dominated computer science was an approach that was *not* shared by most working programmers. As historian Nathan Ensmenger has argued, computer programming was generally regarded as an undisciplined and unscientific activity throughout the 1950s. By the early 1960s, computer scientists and professional programmers looked upon each other in a mutually suspicious manner. Ensmenger has argued that "computer scientists expressed disdain for professional programmers, and professional programmers responded by accusing computer science of being overly abstract or irrelevant".[13] Consequently, programming languages designed by computer scientists would not always be understood as injecting virtuous deeds and rules by professional programmers, but rather would be seen as impractical and cumbersome. In contrast, programming languages designed by industrial researchers were looked upon by equal suspicion by computer scientists, who argued they were inconsistent and nothing but short hacks. Programming languages were designed by scientists aiming at universality, but also by industrial researchers wanting efficient coding for particular machines and by local hacks making local machinery more approachable. Programming languages were really something that came about in widely different ways, with widely different objectives.

The different approaches to programming languages, and programming in general, were reflected in their first popular incarnations. The first widely known programming language was Fortran, developed by IBM and standardised in 1957. It was soon followed by the first Algol version in 1958, which was standardised in 1960.[14] Cobol, issued by the US Department of Defense followed suit in 1962, developed jointly by a committee of European and American scientists.[15] While Fortran and Cobol succeeded as widely used programming languages, and are still used in some quarters, Algol was less successful in practice but became important scientifically.[16] Of the early languages, the universality and formalistic virtue wished for by Gorn, Brown and Carr in 1954 were most evident in the Algol programming language, created by high-profile computer scientists like the

[13] Ensmenger, *The computer boys take over : computers, programmers, and the politics of technical expertise*, 129.
[14] On Fortran, see *IEEE Annals of the History of Computing*. vol. 6 (1984). On Algol, see David Nofre, "Unraveling Algol: US, Europe, and the Creation of a Programming Language", *IEEE Annals of the History of Computing* 32, no. 2 (2010).
[15] On Cobol, see *IEEE Annals of the History of Computing*. vol. 7 (1985).
[16] Mark Priestley has argued that Algol became a paradigmatic example within computer science and computer language design, influencing technical and organisational decisions throughout the 1960s and the early 1970s. See Priestley, "Logic and the development of programming languages, 1930 - 1975".

Dutch Edsger W. Dijkstra, the Dane Peter Naur and the American John W. Backus.[17] With Algol, a transatlantic committee put their heads together to create a programming language, independent of any manufacturer, documented and specified so that it could be used on a wide variety of machinery and mathematical in its form, so that it could express almost any computable algorithm. The objectives behind Cobol had some resonance with this, as it was deliberately constructed in a way that encouraged portability from one type of machine to another, even though the implementers struggled to meet this criterion. Cobol was also a deliberate effort to make a programming language English-like and readable, in stark contrast to the mathematical virtuosity put into Algol. Algol cohort Edsger W. Dijkstra even wrote that Cobol "cripples the mind" because of what he perceived as its linguistic ugliness and inconsistencies.[18] The different perceptions of what constituted good programming and software development were obvious: the humanly readable Cobol and the formalism of Algol represented different approaches to achieve virtuous programming, and would typically appeal to different communities of computing. Obviously, other differences between the two approaches existed, like their different national and institutional origins, intended application domains and the level of commercial and industrial support. These differences were no less important than how the languages reflected different approaches to programming. For reasons of clarity it is, however, sufficient to argue that the basic two-pronged categorisation could be upheld regardless of the demarcation criterion, although some reservations about this must be maintained.[19]

Throughout the 1960s, a whole range of new programming languages appeared, some from the halls of computer science, some from the computer industry. The development of hundreds, or even thousands, of programming languages created a situation comparable to the Tower of Babel: confusion caused by a scattering of numerous languages.[20] Once again, calls for universality and machine independence appeared, similar to those initial concerns in the 1950s. At the end of the 1960s, scientific communities as well as the computer industry strived toward the creation of a single,

---

[17] J. Perlis Alan, "The American side of the development of Algol", *SIGPLAN Not.* 13, no. 8 (1978); Naur Peter, "The European side of the last phase of the development of ALGOL 60", *SIGPLAN Not.* 13, no. 8 (1978).

[18] Here quoted from Ensmenger, *The computer boys take over : computers, programmers, and the politics of technical expertise*, 100.

[19] In particular, the popular view that Algol and its formalism were particularly European has been challenged recently. See Nofre, "Unraveling Algol: US, Europe, and the Creation of a Programming Language".

[20] Jean Sammett's landmark survey from 1969 reviewed or described about 120 languages. See Sammet, *Programming languages: history and fundamentals*.

powerful, programming language to cater for as wide a range of applications as possible, much like a programmer's lingua franca. IBM's PL/1 and new versions of Algol (Algol 68) were created with this in mind. However, both failed to replace the "twin towers" of Fortran and Cobol – universalism prevailed by other means than PL/1 and Algol68, but by sheer path dependence.[21]

Considering systems programming, which involves software that is designed to operate the hardware and to provide a platform on which to run application software, the situation was somewhat similar. Here, the efficiency of the compiled code was of even greater importance, and the idea of one particular all-embracing language like PL/1 and Algol 68 was not viable. However, the design of languages for system implementation was still influenced by the idea of aiming at higher levels of abstraction, and the ideal of one systems implementation language that could replace a whole slew of others gained support.

As a result, a hybrid entity of so-called "machine-oriented higher level languages" was introduced when the 1960s turned into the 1970s.[22] The goal was to achieve almost assembler-level performance as well as a high level of abstraction. One of the first examples of this was PL/360, created by the Swiss computer scientist Niklaus Wirth in 1968. Wirth would later on greatly influence the history of programming languages more in general, when he created the programming language Pascal.[23] However, PL/360 would influence the sub-discipline of systems programming language design, a field that was also close to the telecommunication industry, which also relied on highly efficient code in its systems.

[21] Ceruzzi, *A history of modern computing*, 107.

[22] See W. L. van der Poel, L. A. Maarssen, and International Federation for Information Processing. Technical Committee 2., *Machine oriented higher level languages : proceedings of the IFIP Working Conference on Machine Oriented Higher Level Languages, Trondheim, Norway, August 27-31, 1973* (Amsterdam; New York: North-Holland Pub. Co. ; American Elsevier, 1974).

[23] Niklaus Wirth, "A Brief History of Software Engineering", *IEEE Annals of the History of Computing* 30, no. 3 (2008).

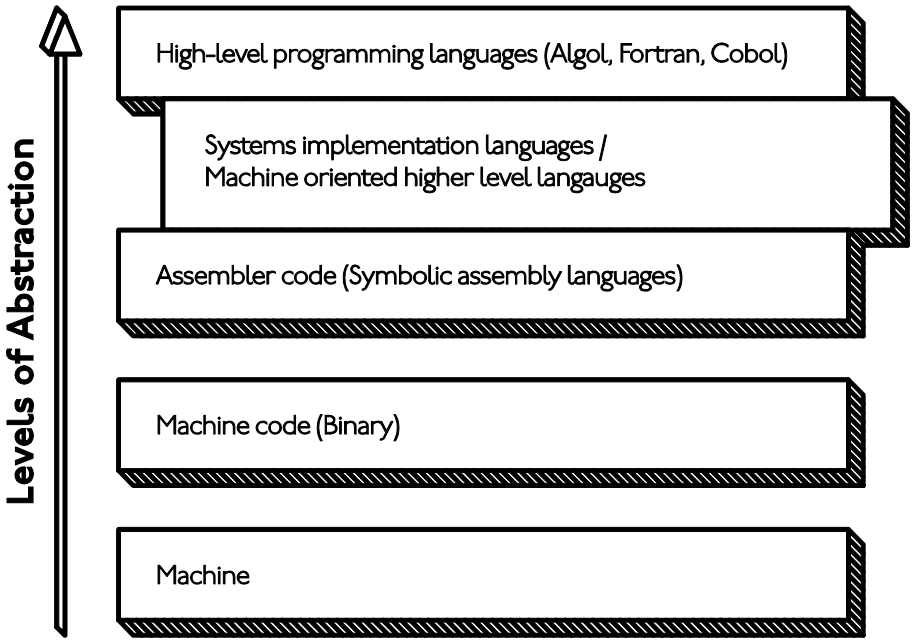

**Figure 2.2 Machine-oriented higher-level languages**

During the early 1970s, the balance between abstraction and performance was sought in development along the lines of "machine-oriented higher-level languages," which provided abstract constructs that would easily compile into effective machine code. Another important change in the development of software and the design of programming languages came about at the end of the 1960s, with the emergence of the hotly contested field of software engineering. This would also influence programming language design throughout the 1970s.

## Software engineering

Programming has always seemed to play catch-up on the advances in hardware. The technical, managerial and organisational challenges related to programming seemed to be in a constant turmoil, inspiring widespread discussions on the roots of the problem. Typically, these discussions would be wrapped up in the concept of the software crisis, a term used to describe the various troubles with software, as in always being late, over budget and below expectations. According to the historian Nathan Ensmenger, the years between 1968 and 1972 were a major turning point in the history of programming and the understanding of its problems, as "the existence of a looming software crisis [was] widely and enthusiastically embraced within

the popular and industry literature".[24] The problems and one proposed solution came to a head in 1968, when it was the subject of a Nato-sponsored conference held in Garmisch, an event that has taken on almost mythical proportions in terms of importance in the literature.[25] The conference introduced the term "software crisis" to describe the problems related to the fact that software was almost "[…] never produced on time, never meets specification, and always exceeds its estimated cost".[26] Secondly, the conference proposed a cure. The term "software engineering" was brought up and according to the organisers it implied "the need for software manufacture to be based on the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering".[27] Various interpretations of what this software engineering concept should consist of and which established engineering concepts it should be based on have been a dominant strand in debates in the software field ever since.[28]

The very conference was, at least to some of its participants, a continuation of the early Algol effort. One of the organisers was Friedrich L. Bauer, which had held a prominent position in the Algol programming language effort. To Bauer and his compatriots at the Nato conference, the solution to the software crisis was closely aligned with the mathematical discipline they had strived for in programming language design. They basically moved the formalistic development virtue from the scientific approach of designing programming languages to a similar approach to the art of creating software.

---

[24] Ensmenger, *The computer boys take over : computers, programmers, and the politics of technical expertise*, 195.

[25] Over the years, this conference has been used in a number of historical accounts. In particular, see Donald MacKenzie, "A View from the Sonnenbichl: On the Historical Sociology of Software and System Dependendability", in *History of Computing: Software Issues*, ed. Reinhard Keil-Salwik Ulf Hashagen, Arthur L. Norberg (Berlin: Springer, 2002); MacKenzie, *Mechanizing proof : computing, risk, and trust*; Mahoney, "Software as Science - Science as Software"; ———, "Finding a history for software engineering". See also the proceedings of the conference, "Software Engineering", (Garmisch, Germany, 7 - 11 October 1968).

[26] The discussions about the terminology are reported in the proceedings, see "Software Engineering", 119 - 25.

[27] Ibid., 13.

[28] Michael S. Mahoney's search for a history of software engineering is made along the lines drawn up at the Garmisch conference. He looks at how various competing agendas for the emerging software engineering profession were essentially tied in with managerial, social and political ideals and furthermore, how these were tied in with models of other engineering professions. According to Mahoney, mechanical engineering, applied science and industrial engineering were evoked to shape software engineering. See Mahoney, "Finding a history for software engineering".

Another important contribution of the Nato conference was an increased attention to concepts that could facilitate modularisation in the software development process, ranging from structured programming approaches to modularisation techniques. Modular techniques in software development meant that one piece of programming code could be written with little knowledge of the code in another module, and secondly, techniques that allowed "[…] modules to be reassembled and replaced without reassembly of the whole system".[29] Finding solutions to the dilemma of "mathematical rigor for small programs against the intractability of large programs" were high on the agenda.[30] Whereas the rigour that Bauer, Dijkstra and their likes idealised was tractable when approaching small programs, it was all the more difficult to achieve when working on large and complex systems. Nevertheless, "mathematical rigour" was still the guiding principle for a large community of technical and scientific experts interested in programming.

This would soon feed back into the subject of programming language design, making the development virtue go full circle. While the issue of abstractions was prominent in the 1960s, concepts for facilitating modularisation in programming language design were actively sought in the 1970s, following up the issues at hand when building the software engineering discipline.[31] As such, it was no coincidence when Jack B. Dennis of MIT lectured on modularity in an advanced course in software engineering in the winter of 1972 – and that one of the issues that he approached was how existing programming languages lacked facilities for "proper" modularisation.[32] The advanced course was a follow up of the Nato-sponsored events of the late 1960s, and organised by Friedrich L. Bauer, the chairman of the 1968 conference at Garmisch.[33] As such, Dennis's concern with modularity tied in with Bauer's prior interest in structured programming and mathematical virtuosity – and how this should be tied in with programming language design and software engineering.[34]

According to a recent study on the relationship between software engineering and programming language research by Barbara G. Ryder, Mary Lou Soffa and Margaret Burnett, "Software engineering research and

---

[29] David Lorge Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", *Commun. ACM* 15, no. 12 (1972): 1053.
[30] Wirth, "A Brief History of Software Engineering".
[31] Ibid.
[32] Jack B. Dennis, "Modularity", in *Advanced Course on Software Engineering*, ed. F. L. Bauer (Berlin: Springer-Verlag, 1973).
[33] The lectures are compiled in F. L. Bauer, ed. *Advanced Course on Software Engineering*, Lecture Notes in Economics and Mathematical Systems (Berlin: Springer-Verlag,1973).
[34] Dennis, "Modularity".

programming language design have enjoyed a symbiotic relationship, with traceable impacts since the 1970s, when these areas were first distinguished from one another." [35] This symbiotic relationship was found to have influenced several major features of modern programming languages, like data abstractions and modularity concepts. This relationship first necessitated a separation of the two fields, a separation the study traces to the early 1970s and the advent of two separate conferences: the first Symposia on Principles of Programming Languages, held in 1973, and the first International Conference on Software Engineering, held in 1975.

Summing up, two particular concerns dominated the early discussions and developments within computer science and the computer industry when entering the 1970s: abstractions and modularisation. While the first concern was evident in high-level languages that appeared in the late 1950s, the latter emerged as a greater concern during the 1970s. The former was mainly related to the move from scientific computing towards real world applications that took place during the 1960s. The latter was related to the influence of the emerging software engineering discipline, as indicated and initiated at the Garmisch conference. Both changes were shaped by technical communities that valued the mathematical development virtue, in particular the participants in the community of computer scientists.

## Programming switches

In May 1965, after seven years of intensive research and development, the American AT&T introduced the very first software-controlled computer in telecommunication switching by putting their "Number One Electronic Switching System" (No. 1 ESS) into service.[36] By 1975, over 800 switching systems controlled by electronic computers were in operation worldwide, and numerous telecommunication administrations were planning on introducing computer-controlled switching in their networks over the next few years.[37] During this 10-year period large electromechanical switches were being replaced by switches that, essentially, were software programs – switches became programs. The production and development of this equipment was very different to the way electromechanical switches were put together. Balls of wires were increasingly replaced by loops of programming code at the development and production facilities of many a

[35] Ryder, Soffa, and Burnett, "The Impact of Software Engineering Research on Modern Programming Languages": 431.

[36] No. 1 ESS is analysed in numerous sources. See for example the articles in Bell System Technical Journal, 43:5, September 1964. The Bell System Technical Journal is available in its entirety online. See http:// bstj.bell-labs.com/

[37] Hills and Kano, *Programming electronic switching systems - real-time aspects and their language implications*.

telecommunication manufacturer, and computer programmers were hired at a great rate.

This was a radical technological change where software were introduced into the very centre of telecommunication, as switches and exchanges are the hubs in the communication networks, connecting telephone calls and providing the transfer of speech from one phone to another. By introducing computer-controlled switching and later digital transmission of speech and communication into the networks, the efficiency and potentiality of network operations increased dramatically.

Computer-controlled switching systems evolved considerably in the 10 years between the first public uses of the technique in 1965. By the mid-1970s, AT&T was rolling out their fourth incarnation of their ESS series. Challengers like L. M. Ericsson, ITT, Siemens, Northern Telecom, NTT and Philips were all steaming ahead developing their own computerised switches on a large scale by this time, and only a few years later, fully digital switching saw the light of day.[38]

Programming a telecommunication switch was considerably different to programming applications for a general computer. The general development of programming languages, towards higher levels of abstraction and concepts that would facilitate higher degrees of modularity was not immediately of use to those grappling with the very first computer-controlled switches in the late 1960s. It was strongly believed that the trade off between machine performance and coding efficiency was too great when using general high-level languages.

However, other objectives valued in telecommunications were more compatible with high-level languages, in particular the idea that high-level languages could foster more reliable programming. If there was one common ideal in telecommunications, it was the idea of the telecommunication system as critical, needing high reliability and availability. While operations of a general computer may be stopped without any serious consequences, telecommunication systems relied on “service continuity”, meaning that few interruptions were tolerated.[39] The standard applied for telecommunication systems at the time was that a two-hour break in the operation of an exchange was tolerated during its expected 40 years of service, something

[38] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 283-90.

[39] The difference between general computing and computing in switching systems was already highlighted in the first description of the pioneering No. 1 ESS by the AT&T. See W. Keister, R. W. Ketchledge, and H. E. Vaughan, "No. 1 ESS: System Organization and Objectives", *Bell System technical Journal* 43, no. 5 (1964).

completely different to the fast changing world of computing.[40] These availability standards would necessitate extremely reliable software, which again was understood as a good reason to use high-level languages and software engineering practices.

Both concerns, the low efficiency of code from high-level programming languages and the higher reliability of coding done with high-level languages opened up for a new hybrid or boundary object, so-called machine-oriented languages and even more specifically, telecommunication-oriented languages.[41] Both were touted as the solution to the mounting programming troubles at telecommunication manufacturers related to the development of software. The main idea was to combine the general knowledge of high-level programming languages and the reliability it could enforce with that of efficient utilisation of computing resources that lower level coding could provide.

The problem associated with programming large telecommunication switches attracted telecommunication engineers, computer scientists and computer programmers, many of them interested in designing these specific programming languages. These people formed a distinct community of technological practitioners in the cracks between engineering and science, telecommunications and computing. From the late 1960s, this community gathered around organisational novelties like conferences on "Software Engineering for Telecommunication Switching Systems", which were first held in 1973 (SETSS), and within tracks in already established channels, like the International Switching Symposium, and eventually within the technical wing of the ITU.[42]

Up until the mid-1970s, the programming of computer-controlled systems was, with few exceptions, done with assembly-level languages, but the development of machine-oriented and telecommunication-oriented languages was in the making. This could look like a local tower of Babel: in a widely read survey conducted by M. T. Hills at the University of Essex and S. Kano from the Japanese NTT (the Japanese telecommunication administration) in 1975, 25 different programming languages were considered suitable for programming switching systems, and several

---

[40] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 262.

[41] See, for example, the Poel, Maarssen, and International Federation for Information Processing. Technical Committee 2., *Machine oriented higher level languages : proceedings of the IFIP Working Conference on Machine Oriented Higher Level Languages, Trondheim, Norway, August 27-31, 1973*.

[42] See, for example, the proceedings of the first SETSS conference: "Software Engineering for Telecommunication Switching Systems", (Stevenage, 2 - 5 April 1973).

machine-oriented, real-time and specifically telecommunication-oriented languages were analysed in detail.[43]

In the Bell System, assembly languages were used up until the more evolved EPL programming language was introduced when the fourth incarnation of their ESS switches was developed.[44] The Canadian Bell Northern used their own Protel programming language for the DMS family of switches. In France, the research institute CNET (the French National Center for Telecommunication Research) had created Pape (also named LP2), a programming language that was used for most exchanges created by various French manufacturers. Pape was tailor-made for programming telecommunication equipment, as part of what was to become a future Alcatel switching system.[45]

The Swedish company L. M. Ericsson had created their own programming language Plex, which was used throughout the 1970s and 1980s in their Axe series of switches.[46] In Japan, the administration NTT had developed a programming language called DPL, which was used to program early computerised switches produced by NTT's preferred manufacturers NEC, Hitachi, Oki and Fujitsu.[47] Finally, the European companies in the ITT conglomerate used a programming language called ESPL/1, which in some respects looked like IBM's PL/1 language. All of the manufacturers augmented their programming language with the possibility of inserting assembly and machine code, and was as such not a complete replacement of their old ways.

---

[43] A survey of this is given in Hills and Kano, *Programming electronic switching systems - real-time aspects and their language implications*.

[44] Later on, the programming language C became the standard when programming telecommunication equipment in the Bell system. For an overview, see Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 283-89.

[45] M. Martin, "Utilization of the high level language Pape for the E12 siwtching system software", in *Third International Conference on Software Engineering for telecommunciation Switching Systems* (Helsinki, Finland: Institution of Electrical Engineers, 1978).

[46] Plex is described in Göran Hemdal, "AXE 10 - Software Structure and Features", *Ericsson Review* 53, no. 2 (1976). On Axe, see also Mats Fridlund, "Switching Relations and Trajectories: The Development Procurement of the Swedish AXE Switching Technology", in *Public Technology Procurement and Innovation*, ed. Charles Edquist, Leif Hommen, and Lena Tsipouri, *Economics of Science, Technology and Innovation* (Norwell, Mass.: Kluwer Academic Publishers, 1999).

[47] Details about DPL and the early Japanese computer switches are found in Shinji Takamura et al., *Software design for electronic switching systems*, IEE telecommunications series 8 (Stevenage Eng. ; New York: P. Peregrinus on behalf of the Institution of Electrical Engineers, 1979).

Did these early examples of firm-specific programming languages bear any resemblance to programming languages that originated in computer science? Technical inspiration from computer science in terms of generalities was obvious in most of these languages. For example, IBM's PL/1 programming language was a main source of inspiration, in particular at firms like the ITT and the NTT.[48] Furthermore, in terms of governance, all these programming languages were considered the property of the telecommunication firms, in contrast to the commons approach typical to many computer science programming languages.[49]

## Interfaces and interaction

Up until the late 1960s, little interaction existed between those interested in telecommunication and those concerned with general computer systems. They were parallel worlds, separated by high walls, technologically, culturally and organisationally. By the early 1970s, this changed, as more manufacturers developed computer-controlled switching systems and computer networks and interactive systems became of great interest to the computer specialists.

Furthermore, a new "family" of programming languages specialised for the use in telecommunications appeared, facilitating interactions between communities interested in programming language design and those working on switching systems. The challenges of programming telecommunication switches attracted various groups and communities, and interfaces between established communities of computer practitioners and telecommunication engineers emerged gradually.

The Nato conferences on software engineering offers a glimpse of the complex composition of the international computer and software community at that time, and more implicitly a way to understand the interactions (or the lack of such) with the international telecommunication community and industry.[50] As the editors of the conference report made clear, the Garmisch conference was special in the way it gathered both practitioners and academics:

[48] See for example the various contributions of the NTT and the ITT to the CCITT organised Team of Experts, detailed out in chapter three in this thesis.
[49] Algol is the most obvious example.
[50] "Software Engineering"; "Software Engineering Techniques", (Rome, 27 - 31 October 1969).

> The Garmisch conference was notable for the range of interests and experience represented amongst its participants. In fact the complete spectrum, from the inhabitants of ivory-towered academe to people who were right on the firing line, being involved in the direction of really large scale software projects, was well covered.[51]

However, none of the groups was homogenous. Some represented large computer firms such as IBM, while others general electronics firms like AEG. This was more obvious at the 1969 conference, where companies such as Boeing, Siemens and General Electric also were present.[52] The conference was an invitation-only meeting, gathered together by the main organisers' of the event, the German computer scientist Fritz Bauer, H. J. Helms, a Danish member of Nato's science department, and the French computer scientist Louis Bolliet.[53] Few participants from the established telecommunication industry were present at both conferences. At the meetings in Garmisch and in Rome, representatives of the American Bell Telephone Laboratories were participating, but that was about it for the communication industry.[54] As Bell was the pioneering firm in terms of computer-controlled switching systems, with the introduction of the computer-controlled Number 1 Electronic Switching System (No. 1 ESS) in 1965, their participation was no surprise.[55] At the 1968 conference, the software development of Bell's No. 1 ESS was presented as an example of the particular difficulties faced when programming large, complex and real-time systems.[56]

While the two Nato-sponsored conferences are considered watershed events within the history of software, no similar event has a similar status within the history of telecommunications. Two examples still illustrate its emergence. In 1966, the very first International Switching Symposium (ISS) in Paris was marked by the appearance of the first lectures on software used

---

[51] "Software Engineering", 7.

[52] It should be noted that General Electric was at the time one of the major computer manufacturers as well as part of the general electric industry. See for example H.R. Oldfield, "General Electric enters the computer business-revisited", *IEEE Annals of the History of Computing* 17, no. 4 (1995).

[53] The information on the organizing of the Garmisch conference is found in MacKenzie, *Mechanizing proof : computing, risk, and trust*, 34 - 35.

[54] Some of the participants from the general electronics industry were involved with telecommunications, like Dr. F. Hofman from Siemens, an attendee of the 1969 conference. However, this is impossible to tell from the details available.

[55] At the 1968 Garmisch conference, Dr. E. E. David, Dr. M. D. McIllroy and Mr. J. A. Harr from Bell Labs participated. At the 1969 Rome conference, Dr. W. S. Brown, Dr. E. E. David and Dr. W. Ulrich from Bell Labs participated.

[56] Peter Naur et al., *Software engineering : concepts and techniques : proceedings of the NATO conferences* (New York: Petrocelli/Charter, 1976).

for programming telecommunication switches.[57] A large conference arranged by the Institution of Electrical Engineers (IEE, a British professional organisation, now a part of the Institution of Engineering and Technology (IET)) on switching techniques for telecommunication networks was held in London in 1969, which also included presentations about software-specific problems when creating computer-controlled switching systems.[58]

While a large number of computer scientists at various academic and research institutions were present at the Nato-sponsored conferences on software engineering, the IEE conference on switching drew participants almost exclusively from industry and various telecommunication administrations-run organisations.[59] The issues discussed, however, had some overlap with those at the software engineering conferences. One example was one session devoted to various aspects of Bell's No. 1 ESS.[60] One paper touched on the very essence on the software crisis, but within a telecommunications project. The problems associated with delivering the software necessary for the pioneering ESS system on time and according to expectations were presented in some detail in a paper by E. Earle Vaughan.[61] Another mutual concern was discussed in a session on "Control and Software",[62] where two of the papers were discussing programming techniques and languages in a telecommunication context.[63] In this session, one of the few academia-affiliated participants was to be found: M. T. Hills and H. Constantine from the University of Essex, which was an important academic partner to the British telecommunication administration at the time and would organise special conferences about software in telecommunication systems.

These two conferences marked the beginning of a period where software rose in importance for both administrations and manufacturers. Four years later, this was all the more obvious, as the IEE considered software engineering for telecommunications systems worthy of a conference on its own: Software Engineering for Telecommunication

---

[57] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 265.

[58] "Switching Techniques for Telecommunication Networks", (London, 1969).

[59] While the reports from the Nato software engineering conferences offer lists of participants, the IEE conference only lists paper authors in its proceedings. 103 papers and 129 individual authors are listed. See Ibid.

[60] Eight papers were presented. See Ibid., 447 – 75.

[61] E. Earle Vaughan, "Development history of No. 1 ESS - Software", in *Switching Techniques for Telecommunication Networks* (London, 1969).

[62] "Switching Techniques for Telecommunication Networks", 190 - 212.

[63] Hills and Kano, *Programming electronic switching systems - real-time aspects and their language implications*.

Switching Systems (SETSS) was held for the first time in 1973 at the University of Essex, with 64 authors presenting 36 papers, all on different aspects of software in telecommunications.[64] This change illustrates the growing importance and interest in software in the telecommunication industries, as well as how this coincides with the maturation of computer science and emergence of software engineering.

SETSS was not the only intersection or contact zone for interaction between software and communications. A parallel arena for discussion, knowledge sharing and creation on software systems similar to those employed in telecommunications, grew out of the programming language community within the International Federation for Information Processing (IFIP), which was set up under the auspices of UNESCO in 1960.[65] A conference on so-called machine-oriented languages was held in the Norwegian city of Trondheim in 1973, which proved to be the starting point of a particular working group within IFIP working on machine-oriented higher-level languages, the 2.4 working group.[66] This is the subject of the next section.

## Closer to the machine

Contact zones like transnational working groups or international conferences do not appear by themselves. Sometimes they emerge due to the mobility of people. The 1973 conference on machine-oriented higher-level languages and the IFIP 2.4 working group can be attributed to one such move: in the winter of 1970, the American Mark Rain arrived in Trondheim, Norway.[67] Previously, he had worked as a programmer at the American computer manufacturer Burroughs, a company particularly known for using a particular subset of Algol as their main programming language. In 1970, Rain got a position at Sintef, the foundation of scientific and industrial research at the Norwegian Institute of Technology (today named NTNU, the Norwegian University of Science and Technology), where he started working on the design of a machine-oriented language, later to be named

[64] "Software Engineering for Telecommunication Switching Systems".

[65] On the history of IFIP, see Ksenia Tatarchenko, "Cold War Origins of the International Federation for Information Processing", *IEEE Annals of the History of Computing* 32, no. 2 (2010).

[66] Poel, Maarssen, and International Federation for Information Processing. Technical Committee 2., *Machine oriented higher level languages : proceedings of the IFIP Working Conference on Machine Oriented Higher Level Languages, Trondheim, Norway, August 27-31, 1973*.

[67] The following details are found in the transcriptions of a panel session at the IFIP conference. See Ibid., 400 - 01.

Mary.[68] According to Rain, his interest in this had already begun at Burroughs, but it was still very much an undefined field with only some vague predecessors:

> At the time when I came to Norway and started with the MARY project, in the winter of 1970, PL/360 had been published. There were rumours that there were a language called PASCAL but I could not find any reference to it. BLISS had just been published, and there was some interest in what I am calling machine-oriented languages, certainly it was nebulous and no one knew what was going on or how.[69]

By 1972, Rain had set up a printed bulletin, which drew more than 500 subscribers around the world. By 1973, the nebulous conditions would drift away and clarity would swathe the field at the conference on so-called machine-oriented languages, held in Trondheim. According to Rain, it seemed that "we have tapped off a running wave that nobody really knew existed".[70] The running wave was partly formed around problems related to how one could programme efficient code, yet still retain the readability of high-level programming languages. At first, such an intersection between machine orientation and machine independence seemed to be quite impossible. However, and this would become the premise for the future interest in machine-oriented higher level languages, certain machine-independent tasks could be approached in a machine-oriented manner, reaching the machine a little better than what was deemed possible in high-level languages of the ordinary kind. The interest was, consequently, to figure out whether these tasks could be programmed close to the machine, by applying general principles. If so, one could approach the problem of systems programming in a more or less machine-independent way.

To many of the participants at the 1973 conference, these aims were coupled with the question of so-called "portability", which implies that a machine-oriented language should "make possible transportation of

---

[68] On Mary, see Mark Rain, "Some formal language aspects of Mary or Algol X revistited", *Algol Bulletin* 1972.

[69] Poel, Maarssen, and International Federation for Information Processing. Technical Committee 2., *Machine oriented higher level languages : proceedings of the IFIP Working Conference on Machine Oriented Higher Level Languages, Trondheim, Norway, August 27-31, 1973*, 400. PL/360 was Niklaus Wirth's groundbreaking implementation and extension of the PL/1 programming language. Pascal was his later effort on creating a high level language. Bliss was a system programming language created at Carnegie-Mellon for use in writing compiler and operating systems for the PDP-10 computer created by the computer scientist William Allan Wulf.

[70] The full session is transcribed in Ibid., 209 - 26.

programs between different machines and diverse configuration of the same machine".[71] This ideal of portability was very much the same that was brought into the Algol effort, which basically was to free software from the machine. However, when considering systems implementation, which was the most obvious target for the use of such machine-oriented languages, the problems were far greater then when considering the portability of other types of applications.

The Trondheim conference in 1973 was dominated by academia, but nevertheless drew participants from various large computer companies. Indeed, even a separate panel discussion on the industry's views of "the MOL [Machine-Oriented Languages]-problem" were arranged with discussants from Xerox, Univac, IBM and the small Swedish computer manufacturer SAAB Scandia.[72] To the academically oriented participants, great interest was shown in how to combine the insights won through computer science with the question of providing portability at a level close to the machine.

To the participants from the industry, the ideals that were upheld by the academic participants at the conference were not in line with their own priorities: Troost, a manager at the American computer manufacturer Univac and head of their internal programming language developments, was particularly harsh in his condemnation of the academics' interests: "What I have heard you talk about are toys, not tools."[73] To Troost, the issues of portability were of little interest. "The points that other people have stressed as of high interest frequently have a low interest for us. […] Portability is nice, but again it is not that important. From our point of view, the most important thing is the ease of maintenance."[74] The maintenance question would greatly divide the approaches found within academically oriented researchers and those working within industrial firms, as all the panellists in the industry panel agreed that their systems programming language would be an in-house language, and not released to a wider community, mainly because of the fear of costs related to the maintenance of a programming language.[75]

Was the telecommunication community as alienated from the scientific computing field as the computer manufacturers represented at the

---

[71] See William A. Wulf's introduction in Ibid., 7-18.
[72] SAAB Scandia was better known as a car and bus manufacturer.
[73] Poel, Maarssen, and International Federation for Information Processing. Technical Committee 2., *Machine oriented higher level languages : proceedings of the IFIP Working Conference on Machine Oriented Higher Level Languages, Trondheim, Norway, August 27-31, 1973*, 212 - 13.
[74] Ibid., 213.
[75] Ibid., 216 - 17.

IFIP conference? None of the presented papers at the IFIP conference dealt with the particularities in programming telecommunication switching systems. However, before Mark Rain participated and helped organise the IFIP conference in Trondheim, he participated in the aforementioned SETSS conference. While Rain was a typical participator at the IFIP conference, he was one of the few academics at SETTS.

Nevertheless, Rain presented Mary to an audience where it would seem tailor-made, as it was a machine-oriented language with a goal of portability – which greatly could enhance the possibility to move software from one switching system to another. To Rain, that would make it a viable alternative to a series of both high-level languages and macro-languages used in a wide range of switching systems. Did it resonate? Programming languages, administration and maintenance of software are some examples of the issues dealt with at the conference, and the problem Rain was concerned with was both explicitly and implicitly discussed in numerous papers.[76] One issue that dominated was that of programming languages, and in particular the challenges in using high-level languages in an application area where performance and reliability were of the essence. Out of 36 presentations, 14 papers dealt with language specific problems. And out of these, nine papers discussed explicitly the use of either high-level or machine-oriented higher-level languages in the setting of telecommunication switching. Rain was in all likelihood one of the few attendees using the vocabulary of machine-oriented languages, a vocabulary frequently used at the IFIP conference. However, it was clear that programming switching systems was believed too special, that it had other characteristics than that of regular programming.

The special needs when programming telecommunication equipment were not without precedence. Telecommunications shared this in-between role with a number of industries concerned with real-time issues, like those involved with computer control of industrial processes and machinery. Programming languages that combined the machine-oriented features with parallel constructs were first made in the process-control area, pioneering the move discussed at the Trondheim conference. Well-known examples were the programming languages Pearl and Coral66.[77] The communities of technological practitioners interested in the intersection between real-time process control and programming language design had a similar international footing as those discussed above, and in some respect they shared the same arenas. The international discussions on these issues took place primarily within the International Federation of Automatic Control (IFAC) and in

[76] "Software Engineering for Telecommunication Switching Systems".
[77] All three are mentioned in Hills and Kano, *Programming electronic switching systems - real-time aspects and their language implications*.

conjunction with the above-mentioned IFIP. [78] Quite early on, the communities involved in real-time programming languages oriented themselves towards the possibility of international standardisation, as evident from the first international conference on programming languages for machine tools, Prolamat for short, which was held in Rome in 1969. The standardisation efforts were first done through the accommodation of existing high-level languages like Fortran, and later on through concerted efforts towards the development of a so-called Long Term Procedural Language (LTPL), which sprang out of the European branch of the International Purdue Workshop in the early 1970s.[79] Also the International Organization for Standardization (ISO) was involved in the quest for a programming language for machine tools, already from the late 1960s.[80] Also here, the development of specialised and standardised programming languages was seen as a remedy for the mounting problems with software. Theodore J. Williams of the Purdue Laboratory for Applied Industrial Control argued in the early 1970s that:

[78] Examples are the joint IFAC and IFIP workshops on real-time programming and programming languages for numerical control. See for example William Henderson Paterson Leslie, "Numerical control programming languages: proceedings of the 1st International IFIP/IFAC PROLAMAT Conference, Rome 1969" (Amsterdam, 1970); J. Hatvany, "Computer languages for numerical control: proceedings of the Second IFIP/IFAC International Conference on Programming Languages for Machine Tools, PROLAMAT '73,Budapest, April 10-13, 1973" (Amsterdam, 1973); P. D. Griem, "Real time programming 1975: proceedings of the IFAC/IFIP Workshop Boston/Cambridge, Mass" (Oxford, 1976); C. H. Smedema, "Real time programming, 1977: proceedings of the IFAC/IFIP Workshop, Eindhoven, Netherlands, 20-22 June 1977" (Oxford, c1978). The history of IFAC is dealt with in Christopher Bissell, "Control in the technical societies: a brief history.", *Measurement and Control* 43, no. 7 (2010).

[79] The LTPL effort is briefly mentioned in Whitaker, "Ada—the project: the DoD high order language working group". See also M. Kronental et al., "The LTPL-E tasking proposals", *Software: Practice and Experience* 11, no. 1 (1981).

[80] Werner B. Mangold, "N/C Language Standardization in I.S.O", in *The Second IFIP/IFAC International Conference on Programming Languages for Machine Tools, PROLAMAT '73*, ed. J. Hatvany (Budapest: North-Holland Publishing Company, 1973).

> Misjudgements by project personnel concerning project software requirements and capabilities have resulted in a high percentage of late and incomplete computer process control projects. By easing programming requirements through the promotion of use of special high level languages and specific program packages, programming standardization activities promise to greatly ease the above mentioned difficulties.[81]

Many of the efforts in process control and machine tools programming would later on be conflated with the already mentioned large-scale effort towards standardising the programming language Ada.[82] Where the standardisation of programming languages for process control systems was first made within a quite heterogeneous institutional framework, the IFAC was a free-standing international body, the Purdue workshop was a voluntarily organised interest group spun off from a university seminar and the industrial real-time Fortran standard was issued by the Instrument Society of America (ISA), the move towards a standardised programming language in telecommunication involved a rather different kind of authority. To gain momentum in the world of telecommunication, the technical community interested in programming languages for these systems would have to be aligned with the interests of the international telecommunication regime. This is the subject of the next few pages.

## Computing and the international telecommunication regime

At the first International Switching Symposium (ISS) in 1966, some of the first lectures on software used for programming telecommunication switches were presented. ISS was the first intersection between telecommunications and software. At the same venue, the idea of involving the technical flank of the International Telecommunication Union (ITU) in studies on this new subject was launched during informal talks.[83] This meant that the nascent field was considered important to the main organisation in the international telecommunication regime, and something that the organisation could spearhead.

By the 1970s, the ITU was often mentioned as an organisation unlikely to be spearheading anything. ITU's standards were increasingly

---

[81] Theodore J. Williams, "CAM and NC Software Systems: Needs for and Benefits From Generalized and Multi-Industry Standardized Languages", in *The Second IFIP/IFAC International Conference on Programming Languages for Machine Tools, PROLAMAT '73*, ed. J. Hatvany (Budapest: North-Holland Publishing Company, 1973), 1.

[82] See Kronental et al., "The LTPL-E tasking proposals".

[83] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 265.

looking like "hybrid monsters," to use an expression by Raymond Croze, director of the CCITT from 1972.[84] The standards were considered bloated, inconsistent and often arriving too late to make a difference. As digital technology and techniques became more and more applicable in the world of telecommunications in the late 1960s, this tendency could be argued to be all the more visible. While the international computing community was discussing the evolution towards high-level programming languages and the formation of the discipline of software engineering, the ITU was discussing telex tariffs. To be fair, the digital era was gradually making its mark on the ITU from around 1968, as both digital signalling systems and data communication protocols were part of the technical discussions and workings of the CCITT.[85] However, the general impression was of an organisation with considerable difficulties in adjusting to the new realities of digital communications.

What were the people interested in programming languages for telecommunication systems getting themselves into if they were to tag along with the CCITT? Was it the start of a technocratic exercise in technical collaboration or the start of a process to forge the newly won academic and industrial interest in systems implementation languages, so called "machine oriented higher level languages", into a tool for what could be understood as a telecommunication cartel?

Programming was a very different subject from those traditionally covered by CCITT. For a time, there was even doubt to whether the ruling authority of the CCITT, that is its Plenary Assembly, would allow a move that far off the regular path. From 1968, a study group within the CCITT was given the task to report on, among other things, programming languages. The suggestion came from the Swedish telecommunication administration, Televerket. Here, it was seen as important that in the future, the administration could move programs freely between switches made by different manufacturers. In short, they wanted portability. A common programming language was envisioned as an important tool to achieve this.[86] This coincided with a general proactive approach to international standardisation by the Swedish telecommunication administration, in

[84] Raymond Croze used this term on a large number of CCITT recommendations in a speech made to the CCITT Plenary Assembly, the same event where he used the J. G. Thompson article referenced above. See "Minutes of the Plenary Meetings," CCITT Plenary Assembly 6 Orange Book Vol. I - IV 1976, 23, CCITT, ITUA.

[85] A short summary of this is found in Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*.

[86] The background for the Swedish proposal is described in Bertil Forss, in *Autobiographies, "From Computing Machines to IT"* (Stockholm: National Museum of Science and Technology, Sweden, 2007).

particular by its technical director Gösta Lindberg, who together with Bertill Forss was directly responsible for raising the issue of programming languages at the CCITT in 1968.[87] Chapuis and Joel, two close observers of the process, have described it as follows:

> First, its regularly attending delegation of faithful experts had to make room for newcomers, experts in the new discipline of software. This took some time. It is thus hardly surprising if the 1968 – 1972 period was little more than exploration since the CCITT was venturing into virtually unknown waters.[88]

After a period in these unknown waters, the organisation considered moving more actively into the area of programming languages. At the 1974 International Switching Symposium, held in Munich, the chairman of CCITT's 11th study group, Mr. J. S. Ryan, commented on what were perceived as a new role for the CCITT:

> In the past, the International Consultative Committee for Telegraph and Telephone (CCITT) has made only minimal recommendations concerning international switching systems, and very few concerning national signalling and switching systems. […] is this minimal role in signalling and switching adequate in the future? […] It is obvious that the switching engineer whether he is designing international or local exchanges and signalling systems will be directly affected by CCITT Recommendations more in the future than in the past. The question that has been raised is just how far the CCITT can or should go in making Recommendations for national systems and at what point in time.[89]

With the advent of a programming language not only approved by the CCITT, but also created within its ranks, it was obvious that the purpose was to penetrate quite deeply into the switching systems inner workings. In fact, by creating a CCITT recommendation on a programming language, the impression was immediately that telecommunication administrations could impose this tool on manufacturing firms. This caused tensions, which were both an obstacle towards a functional language definition, as well as a necessary frictional element towards a forward-looking language.

While the picture Cowhey paints of the ITU was one where the interests of telecommunication administrations and the many nationally

---

[87] Lindberg also initiated the Nordic participation in Chill as well as the administration's work on data communication standards such as X.21 and X.25. I have previously looked into this in Gard Paulsen, "Samarbeidets protokoll: utviklingen av et nordisk datanett, 1971 - 1981" (G. Paulsen, 2004).

[88] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 265.

[89] J. S. Ryan, "The Role of the ITU and CCITT in telecommunications" in "International Switching Symposium", (Kyoto, October 25-29 1976).

limited manufacturers were in harmony, this was not always the case.[90] A tension between the interests of the administrations and that of various manufacturers was evident in many smaller countries around the world, while in some larger markets the ties were stronger than ever. In the UK, the Post Office directed their favoured manufacturers on their route to digital switching through the System X project throughout the 1970s, and in Japan, the administration, the NTT, applied a so-called coordinated competition system among their favoured manufacturers.[91]

One early indication of the conflict is found in a questionnaire circulated by the CCITT in 1970.[92] This questionnaire tried to gather information on how various administrations looked on the particular challenges of computer-controlled switches, and in particular how the administrations looked on the role of software. 11 administrations replied to the questionnaire. No general consensus existed on how the administrations should organise and act on the new technological reality. The majority of the replying administrations stated that they obviously needed new capabilities to be able to assume responsibility for the management and maintenance of switching systems software. However, one administration in particular held a view that was quite contrary to the others, namely that the administrations should employ a team of specialists that would be responsible for managing and producing programs for new switches on their own. One other reply refers to the question of the separate supply of hardware and software for SPC exchanges. In this case the hardware would be purchased from the telephone equipment manufacturers and the software sub-contracted to software firms. The report given on the questionnaire concludes on this by stating that:

---

[90] Peter Cowhey's conceptualisation of the international telecommunication regime and its national underpinnings was presented in chapter one. See Cowhey, "The international telecommunications regime: the political roots of regimes for high technology".

[91] On System X, see Geoffrey Owen, *From Empire to Europe* (London: Harper Collins, 1999), 282-88. On NTT, see Martin Fransman, *The market and beyond : information technology in Japan* (Cambridge England ; New York: Cambridge University Press, 1990); ———, *Japan's computer and communications industry : the evolution of industrial giants and global competitiveness* (Oxford ; New York: Oxford University Press, 1995).

[92] The findings of this questionnaire are reported in COM XI 1-E (1973 – 1977), CCITT, ITUA. The individual answers are not in the ITU archive.

> To facilitate the work of these specialists, the organizations would be in favour of CCITT recommendations on the presentation, specification and documentation of the hardware and software functions in SPC exchanges. In particular, the majority of them are ready to participate in a CCITT study of an advanced-level programming language.[93]

A questionnaire the following year tried to sound the administrations on what they understood by "an advanced-level programming language". While only seven organisations replied, a considerable difference of opinion was evident on what an advanced-level programming language meant to them. Based on the replies, a high-level language at the level of Algol was wanted by a number of administrations. Some other replies favoured a specification language coupled with advanced assembly techniques. In the previous questionnaire, one responding PTT also argued in favour of using "assembly-level languages in order to not lower the real-time call handling capacity of systems".[94] Regardless of belief in abstraction-based languages, the CCITT reported that there was a general agreement that a standardised high-level language could be justified "on the grounds of maintenance, education of staff, administration and software modification".[95] Others mentioned information transfer, accuracy and readability as reasons to work in this direction. Nevertheless, it coincided nicely with the first SETTS conference (held in 1972) and the mainly academic affair at the IFIP-organised conference on machine-oriented higher-level languages held in Trondheim in 1973.

Would the interest in programming languages among the administrations reinforce the "ancien regime", or would it crack it open? Would it align them with the interests of the manufacturers or could they bank on the support of the computer science community? In this early phase, the CCITT's interest in programming language was unfocused. It was neither dependent on the "ancien regime" nor about to crack it open. It was an indication of the growing concern of a community of technological practitioners interested in the programming of telecommunication switches, a community that were embraced by the CCITT. From about 1973, this unfocused approach changed. A more active and pronounced policy towards programming languages was put into motion by the CCITT when it started a review of a number of existing programming languages, with the intention of elevating the best to the status of a CCITT standard.

[93] COM XI no. 1-E, Annex 4, 33 - 35, CCITT (1973 – 1976), CCITT, ITUA.
[94] COM XI no. 1-E, Annex 4, 35, CCITT (1973 – 1976), CCITT, ITUA.
[95] Ibid., 36.

## State of the art

From 1973, the CCITT was engaged in reviewing a number of existing programming languages, with the intention of elevating the best to the status of a CCITT recommendation. Participants from AT&T, ITT, Siemens, the UK Post Office and the French administration got busy teasing out the strengths and weaknesses of a total of 27 programming languages that had been submitted for review. The review process was based on 15 loosely defined requirements spun around concerns such as machine independence and program portability, the logical structuring of programs, modularisation and extensibility.[96] All in all, the requirements were casually defined and inspired by general tendencies in programming language design at the time, in particular the academic interest in machine independence and program portability and the industrial interest in modularisation were important criteria.[97]

Among the contenders were a host of general programming languages such as Algol 68, Pascal and PL/1, as well as proprietary languages developed by telecommunication manufacturing companies such as ITT and L. M. Ericsson.[98] Quite early on Algol 68 and IBM's attempt on a general programming language, PL/1, were scrapped, supposedly because they were not considered suitable for switching systems.[99] Eventually, seven language were shortlisted as possible candidates: ESPL/1, DPL, Mary, Pascal, Pape, Plex and TPL2. ESPL1, from the international conglomerate ITT and PLEX, from the Swedish company L. M. Ericsson, were two specialist languages with no other aims than that of programming telecommunication switches. Two languages had their origin at technical universities: TPL2 from the University of Essex (created in cooperation with the British Post Office Research Department) and Mary, from the computer centre (RUNIT) at the Norwegian Institute of Technology. Both languages were created with machine orientation, and implicitly efficiency of the compiled code, in mind, and stemmed from the interest towards low-level system implementation

[96] For an overview, see COM XI-No.74-E, Annex E to "Progress report of sub group XI/3-2, High-level programming language for SPC telephone exchanges." COM XI 1973 – 1977, CCITT, ITUA.

[97] "Record of programming exercises and experts' comments" (The "Yellow Document"), COM XI 1973 – 1977, CCITT, ITUA.

[98] The evaluation process is described in the following documents: COM XI-No.74-E, Part IV, "Progress report of sub-group XI/3-2" and COM XI-No.135-E, "Report on the meeting held in Geneva from 18 September to 25 September 1974", CCITT, Period 1973 – 1976, CCITT, ITUA.

[99] COM XI-No. 73-E, "Extracts from the minutes of an informal C.C.I.T.T. meeting held in London from 25 – 27 March 1974", CCITT, Period 1973 – 1976, CCITT, ITUA.

languages that had been a concern of the computer science milieu since the late 1960s.

Pascal, another of the shortlisted programming languages, was a general purpose programming language designed by the Swiss computer scientist Niklaus Wirth, and had been used in various computer platforms since its first publication in 1970.[100] DPL and Pape were languages with an explicit link to telecommunications, created by the Japanese and French telecommunication administrations, respectively.

The reviewers found a number of deficiencies in all the shortlisted languages. In particular, the review panel put great emphasis on the strengths in the specialist programming languages rather than the more generalist languages like Mary and Pascal. However, such technical strengths, due to their particular orientation towards telecommunication switching, were not compatible with the diplomatic nature of the CCITT, where it was deemed impossible to elevate a technology developed and owned by a single manufacturer as to the status of a recommendation. Furthermore, this would have undermined the possibility of using the programming language as a tool for creating telecommunication administrations less dependent on their manufacturers when it came to programming. According to Joel and Chapuis, there "was no way that a language adopted by a given manufacturer could be selected as the sole universal CCITT language".[101]

The working group that was to decide on the question based on the review panel's discussion was put together in such a way that a consensus decision was unlikely. The result was, predictably enough, that: "[…] it was found that it was not possible to obtain a consensus for any one language in either a modified or unmodified form", to quote the original report.[102] Even the two languages that were free of such baggage, Mary and Pascal, were unable to gather enough support. Mary was directly related to the academic interest in machine-oriented higher-level languages and designed by Mark Rain, who, as previously noted, was one of the organisers of the IFIP conference on this subject. Pascal had a similar academic background, designed by Niklaus Wirth, who also had pioneered the move of such implementation languages with his PL360 a few years before the design of Pascal. Pascal ended up being described as not particularly suited to so-

[100] The document COM XI-No-74-E lists ESPL/1, DPL, Mary, Pascal, Pape, Plex and TPL2 as the seven shortlisted languages. The evaluation is reported in Working Party XI/3, Language Descriptions, The "Pink Document", CCITT, ITUA.

[101] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 274.

[102] COM XI-No. 135-E, "Report on the meeting held in Geneva from 18 September to 25 September 1974", CCITT Period 1973 – 1976, CCITT, ITUA.

called “real-time processing”.[103] Wirth did not disagree with this, as he added his own comments to the language description, and found them both “objective and quite accurate”.[104] As such, the ITU participants brushed aside the straight academic route as well as the path laid down by manufacturers. By rejecting the existing, it was up to CCITT to design a new language, something completely new to the organisation and most of its members. This is the subject of the next chapter.

## Some conclusions

What could be gained by opting for a common and standardised programming language? The decision made by the CCITT rested on the strategies of the telecommunication administrations: the idea was first put forward by representatives of the Swedish telecommunication administration in 1968. It was the administrations that were sounded out in the early 1970s, through questionnaires and by common working groups organised by the CCITT. It was their votes that decided that a group of experts should design a new programming language from the ground up in the end of 1974. As such, the decision to study and move forward on the question regarding an international standardised programming language was completely in line with the logics attributed to the international telecommunication regime. Nevertheless, many a manufacturer submitted their favoured programming language to the CCITT review panel, letting outsiders in on one of their ways of managing the complexities of switching software. However, there is little evidence that the manufacturers that submitted their existing programming language altruistically shared their proprietary secrets for the greater good of the technical community. They were simply banking on gaining an advantage if their language could be approved as a CCITT recommendation. In this first round of technical diplomacy and institutionalisation efforts no such advantage was given to any of the manufacturers. The way forward was envisioned along the lines of a new programming language, aligned with the academic interest in machine-oriented higher-level languages. The manufacturers would continue to strive for influence, and some of their representatives would soon be closely aligned to technical communities working together on programming language design.

This chapter has also argued that by the early 1970s, a number of other issues concerning programming coalesced: the ideal of universality through mathematics had grown in legitimacy through the normalisation of computer science. The conceptualisation of software engineering as a

[103] Working Party XI/3, “Language descriptions”, The “Pink Document”, CCITT, Period 1973 – 1976, 95.
[104] Ibid., 91.

solution to the software crisis, influenced by the mathematical development virtue, appeared in the years from 1968 to 1972, and would gradually influence programming language design in the period that followed. The possibility of machine-oriented higher-level languages emerged in the same period. At the same time, computer-controlled switches were coming of age, maturing into a large and extremely demanding field. Together, these developments led to the formation of a distinct community of people interested in programming languages for telecommunication equipment. This community was organised around conferences like the IFIP conference on machine-oriented higher-level languages and the SETSS conferences. In the end, this triggered the interest of participants in the ITU, the main organisation of the international telecommunication regime. Programming languages were boundary objects in which different communities held an interest.

Above, I have claimed that the agendas of these communities were informed by a set of development virtues, ideals about the act of programming. In particular, I have identified how a mathematical oriented development virtue was important to many programming language designers and theorists in the late 1960s. Another agenda was rooted in programming as a way of English-like expressions and more pragmatic in terms of how the act of programming should be managed. Both were understood, by their supporters, as necessary approaches to a set of difficult and complex problems. While the adjective mathematical certainly can be understood as impractical, this was hardly the case here. To be mathematically precise was indeed understood as a virtue, even a practical one.

In the area of systems implementation and machine-oriented higher level languages, two competing development virtues were evident: One was embedded in the agenda promoting portability of machine-oriented languages, an agenda mainly in line with what was considered typical to computer science. The second agenda promoted maintainability and minimal languages, rooted in a more pragmatic development virtue.

I have also highlighted how the virtues of computer science and software engineering influenced telecommunications through a number of intersections and encounters. These existed at an international level, such as the Nato-sponsored conferences on software engineering, conferences organised by IFIP, and more specialised telecommunication conferences such as ISS and SETTS. When CCITT got involved with this, many of these technical issues resonated with the logics of the international telecommunication regime. Could this resonance be extended when approaching a brand new programming language design? This leads us into the next chapter, which is concerned with the first real round of technical diplomacy concerning the design of this programming language.

## 3. Contested designs: agreeing on a programming language for telecommunication

From January 1975, a group of programming language researchers, telecommunication bureaucrats and telecommunication industry experts started to design a programming language from scratch, a language that could be used by telecommunication administrations and manufacturers to programme future equipment. The group, named the High Level Language Team of Specialists, was set up by the CCITT to create this new and special purpose programming language. Five years later, the final recommendation was approved by the CCITT Plenary Assembly and became CCITT Recommendation Z.200.[1]

The programming language was designed within the Team of Specialists until the summer of 1977, when the "Implementors' Forum" replaced the small circle of the specialist group. The initial design was then ready, while the actual implementation and practical tests of the language were the work of the Forum. This chapter is devoted to the Team of Specialists, who they were, where they came from, the ideas and opinions they shared and, most importantly: how did they design the programming language?[2]

In retrospect, the leader of the Team of Specialists has described part of the life of the group as "a period of chaos".[3] Sharp disagreements, conflicts and communication problems characterised much of what went on. Some of the participants even felt that the rifts within the team were too great and that it would be impossible to agree on a coherent language

[1] *CCITT High Level Language (CHILL)*, CCITT Recommendation Z.200 (1980).

[2] The account in this chapter is based on sources from mainly two archives: The ITU archive in Geneva (ITUA) and the private collection of Kristen Rekdal (KRC). Since the Team of Specialists was not part of the official CCITT hierarchy, none of their communications are held by the official ITU archive. However, their reports to the working group XI 3/2 are available. "COM XI-No.19-E," Period 1977 – 1980, CCITT, ITUA, gives a review and a list of the documents. None of the so-called late contributions to the working groups were kept by the ITU. However, the Kristen Rekdal collection holds more complete series of documents from early 1975 and forwards, containing both reports from the Team of Specialists, communication between its members as well as late contributions to the official CCITT process, and the numbered working documents.

[3] Remi Bourgonjon, interview with author, 16 January 2009, Heeze, the Netherlands.

proposal.[4] The differences were mainly about features of the programming language. They appeared as clashes between strongly held ideas about how programming should be done, and how such ideas could be embedded into a programming language. However, it was also a game of strategic positioning involving the team participants' parent organisations, which ranged from manufacturing firms, administrations to research organisations. The design process was one of intense technical diplomacy among specialists coming from a community of technical practitioners as well as delegates strategically positioning the large organisations that employed them. Diplomacy shaped by strongly held virtues as well as industrial strategy was the result.

This chapter delves into issues concerning programming language design, standardisation and diplomatic bickering in detail through a description of the processes. Following this, it answers how the team was able to traverse a number of hurdles and arrive at a coherent proposal. First, I describe the composition of the team, before I pay particular attention to the diplomatic process within Team of Specialists and how, gradually, they were able to align the rather different perspectives of the team participants towards a common proposal. In the end, I arrive at a social network analysis (SNA) of the collaboration and conflicts in the period from January 1975 to the summer of 1977, which I compare to the findings from the qualitative assessments preceding it.

## A Team of Specialists?

The Team of Specialists was unlike CCITT's heavily structured working groups, working parties and study groups. It was an ad-hoc group atypical for the CCITT, in that it was largely autonomous, and it was led by an industry representative: normally in the CCITT, the only ones with voting power were administration representatives. Following this, the group leaders were normally chosen among members with voting power.[5]

At first, the CCITT had hoped to delegate the design job to a single specialist, namely the programming expert Niklaus Wirth.[6] The Zürich-based Wirth had previously designed the programming language Pascal, a hugely

[4] See for example Svein Hallsteinsen, "Reiserapport fra møte i CCITT's spesialistgruppe for høynivå programmeringsspråk, Bern, 26. januar – 5. februar", 13 February 1976, box "CCITT - HLL Team of Specialists - Møtereferater 1974 - 1976, Møte 1-6", KRC. Svein Hallsteinsen was part of a research group at the computing centre of the University of Trondheim (Runit) and sponsored by all the Nordic telecommunication administrations.

[5] The formal working group concerned with the design of a programming language was named XI/3-2, a sub-group of the working party XI/3, which again was a part of the study group XI.

[6] COM XI-No. 135-E, "Report on the meeting held in Geneva from 18 September to 25 September 1974", Part IV, Annex A, CCITT, Period 1973 – 1974, ITUA.

influential programming language created in the late 1960s and popular right up into the 1990s. Pascal was also one of the seven shortlisted languages that the working group considered promising the year before, although it ended up as being deemed as unsatisfactory.[7] While Wirth's best-known contribution to programming language design was deemed unsuitable for the domain CCITT sought, Wirth had practically ignited the field of machine-oriented languages with his creation of the PL360 programming language. PL360 was among the first programming languages to combine exact machine language instructions with features commonly found in high-level languages. Wirth had also been involved in the language review conducted by the CCITT before the decision on creating a programming language. Wirth had consulted Bertill Forss, the Swede who initially had brought the programming language question to the CCITT in 1968, as Forss acted on behalf of the Swiss telecommunication manufacturer Hasler and the Swiss telecommunication administration, as he reviewed some of the existing programming languages as candidates for the CCITT.[8] With his background, Wirth looked like a perfect match for the assignment. However, he declined the offer, after a lengthy process in which the CCITT had tried to facilitate the unprecedented move of trusting an assignment to someone without any formal ties to any telecommunication organisations.[9] The task was then delegated to the Team of Specialists.

On 15 January 1975, a meeting in Bern brought together what was called a fire brigade of international experts. They were to start what had been planned for Wirth.[10] However, just to come to a mutual understanding of what that would involve was no easy task. Meetings in January and February 1975 were held just to agree on a mandate of the Team of Specialists, before its first official meeting could be arranged. Progress came, but often at a numbing speed. Disagreements, tangles and conflicts

---

[7] Working Party XI/3, The "Pink Document", CCITT, Period 1973 – 1976, ITUA, 95.

[8] This is explicated in Forss.

[9] In retrospect, that does not seem very surprising. In a paper by Wirth from 1974, "On the design of programming languages", he lists 11 lessons he had learned as a programming language designer. The last point reads as follows: "Keep the responsibility for the design of the language (and possible changes) confined to a single person." The manacles of the CCITT were perhaps not a natural fit for such an idea. See Niklaus Wirth, "On the Design of Programming Languages" (paper presented at the Information Processing 74, Stockholm, Sweden, August 5-10 1974).

[10] As the report of the meeting explained: "the meeting was a sort of fire brigade action as Prof. Wirth unfortunately was not in a position to carry out the task that the Subgroup had asked of him." See Subgroup XI/3-2, "Report of an informal CCITT meeting held in Bern from 15-17 January 1975", Temporary document No. 4-E, CCITT Study Group XI, Geneva, 10-19 February, ITUA.

came in the way of real technical design. I turn to a report given by a Norwegian delegate, Kristen Rekdal, to illustrate this:

> It was clear from the outset that it was a great deal of disagreements within the group. This seemed to be due to the fact that not everybody was interested in developing a language as soon as possible. Some believed the Team of Specialists should be given a carte blanche and work as fast as possible. Others wanted that XI/3-2 [the superior work group] should be given more control. Furthermore, they wanted a larger and broader Team of Specialists. The Japanese and to some degree the Americans were afraid that the team would be dominated by Europeans.[11]

However, Rekdal hoped that a Team of Specialists would be the right forum to overcome such difficulties. Followed up his description, he added: "By organising the work within a group of specialists that will work more or less full time, I have high hopes that the project will be a success. Normally, it is easier to come to an agreement on professional grounds rather than political."[12] Which "professional grounds" dominated the team? And could such a team be exclusively professional?

The team came to be made up of representatives from the Dutch manufacturer Philips, the NTT of Japan (its administration, with strong influence over the Japanese telecommunication manufacturing industry), the Nordic telecom administrations (which cooperated by sending one common delegate, from a research group at the computing centre of the Norwegian Institute of Technology in Trondheim, Runit), Siemens of Germany, L. M. Ericsson of Sweden, the international manufacturing conglomerate ITT, the British Post Office (then the British post and telecommunication administration) and the Swiss telecommunication administration. Its composition came as much from its ad-hoc basis as from political and industrial strategies, as the team was brought together at a very short notice.[13] Many organisations declined the possibility of being part of the Team. One should note that apart from the representative of L. M. Ericsson, none of the industrial participants came from firms particularly close to the technical leading edge when it came to digital switching. None of them held

---

[11] Kristen Rekdal, "Reiserapport fra møte i CCITT arbeidsgruppe XI/3 om Programmeringsspråk for datamaskinstyrte telefonsentraler, Geneve 10. – 19. februar 1975", 6 March 1975, box "CCITT - HLL Team of Specialists - Møtereferater 1974 - 1976, Møte 1-6", KRC. My translation.

[12] Ibid.

[13] Both the UK Post Office and the French administration participated actively in the review process that preceded the Team of Specialists, but did not actively pursue the work within the Team. Furthermore, many of the delegates of the parent CCITT groups represented organisations not present in the Team.

positions as early adopters of high-level programming languages in their programming of computer-controlled switches.

Compared with similar working groups in the CCITT, the Team of Specialists looked odd. Two comparable groups, one concerned with a so-called man-machine language (called MML) and a specification and description language (called SDL) started out in parallel to the programming language project. The specification language was an effort to standardise the graphical representations of the functionality of telecommunication systems, which was rather different than creating a programming language, which was intended to deliver the functions of the telecommunication systems. Both the MML and the SDL received far more attention from telecommunication manufacturers than the Chill group. Furthermore, the SDL group received more attention from typical CCITT participants, such as electrical engineers and telecommunication experts. However, the very same manufacturer that played a role in the Team of Specialists, L. M. Ericsson, held a high stake in the SDL committee, and just as in the case of programming languages, L.M. Ericsson had pioneered the use of specification languages in the process of designing digital switches.[14]

Who was not part of the Team? Notable absentees were the French as well as the Americans. The representatives of the ITT were, in earnest, represented by their French subsidiary, which sent the Americans Beierle and Parente – indeed a double whammy in terms of making up for absentees.[15] More central French representatives, from the administration or from a larger French manufacturer, were absent. Pioneers like AT&T's Bell Laboratories and the Canadian Bell Northern Research were, as often was the case in international telecommunication cooperation, not participating in CCITT activities in any substantial roles, although the parent study group of the Team of Specialists, Study Group number XI, was led by J. S. Ryan of the Bell Laboratories.[16] None of the actors that participated in the prior language review continued their work in the Team of Specialists in any capacity, with the exception of the participants from the British Post Office (R. T. Boyd).

From spring 1975 until February 1977, the Team of Specialists held seven lengthy meetings at different locations around the world. In between meetings, the group relied on informal correspondence and formal publications of working papers and language proposals submitted by

[14] Hemdal, "AXE 10 - Software Structure and Features".
[15] On the status of ITT, see Sverre A. Christensen, "Switching Relations: The rise and fall of the Norwegian telecom industry" (BI Norwegian School of Management, 2006), 38-45; Robert Sobel, *I.T.T. : the management of opportunity* (New York, N.Y.: Times Books, 1982).
[16] See various reports in COM XI 1973-1977, ITUA.

committee members or by their parent organisations. Details of the locations and dates are summarised in the tables below.

| *Meeting* | *Dates* | *Place* | *Participants* | *Documents* |
|---|---|---|---|---|
| 1 | 28 April - 9 May 1975 | Geneva | 9 | 7 |
| 2 | 25 August - 5 September 1975 | Geneva | 6 | 17 |
| 3 | 8 -19 December 1975 | Bern | 6 | 17 |
| 4 | 26 January - 5 February 1976 | Bern | 7 | 5 |
| 5 | 10 - 21 May 1976 | Bern | 7 | 16 |
| 6 | 12 - 22 October 1976 | Kyoto | 9 | 18 |
| 7 | 21 - 25 February 1977 | London | 8 | - |

**Table 3.1: Meetings in the Team of Specialists.**

Table 3.2 gives a brief overview of who attended the meetings. What seems evident is the apparent stability of the group and the little variation in terms of attendees. In general, the Team of Specialists was dominated by participants that showed up regularly throughout its life.

| | | | *Meeting* | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | Country | Organization | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| J. D. Beierle | France/USA | ITT | X | X | X | X | X | X | X |
| R. Bourgonjon | Netherlands | Philips | X | X | X | X | X | X | X |
| R. T. Boyd | UK | UKPO | X | - | - | - | - | - | X |
| K. Clements | UK | UKPO | - | - | - | - | - | X | X |
| S. Hallsteinsen | Norway | RUNIT | - | - | - | X | - | - | - |
| H. Kvarneby | Sweden | LME | X | X | X | X | X | - | X |
| T. Koizumi | Japan | NEC | X | - | - | X | - | - | - |
| K. Maruyama | Japan | NTT | X | X | X | - | X | X | X |
| K. Rekdal | Norway | RUNIT | X | X | X | - | X | X | X |
| L. Sandberg | Swiss | PTT | - | - | X | X | X | - | - |
| H. Sorgenfrei | Germany | Siemens | X | X | - | X | X | X | X |

**Table 3.2: Participants in the HLL Team of Specialists, ordered alphabetically.**

The mere design of the group heralded difficulties already from the outset. It was in no way homogenous, consisting of bureaucrats representing administrations, researchers as well as representatives of competing manufacturers, such as Philips, the ITT and L.M. Ericsson. The group leader Remi Bourgonjon, from Philips Telecommunications, and the Norwegian researcher Kristen Rekdal were the only group members that could devote

almost their full working time to the project, while the others were participating in addition to their normal work responsibilities within their parent organisations. As a consequence, Bourgonjon and Rekdal held positions that could exert considerable influence over the team's work. However, it was not only the time spent at work in the team that made Bourgonjon and Rekdal important actors in the standardising and design processes. Both had a background in computer science and mathematics and shared a technical expertise and a specialised language not necessarily shared by all the other members of the team. This background was indeed what Rekdal had in mind when he hoped that the professional qualities would overcome the political tensions and make the group special when compared with other CCITT-initiated groups.[17]

Bourgonjon's role as the convenor of the Team of Specialists underscored the peculiar role of this group within the CCITT. As a representative of a manufacturer, Bourgonjon represented a group of participants that held no voting power within the CCITT, and, consequently, was not normally entrusted with group leader roles. Tellingly, Bourgonjon was given the title of convenor rather than chairman. Titles and procedures had a high standing within the CCITT, and the role of convenor indicated the otherness that surrounded the team. Another similar indication was the fact that the first meeting had to be held in Bern rather than in the ITU's native Geneva, which was not a coincidence. The team was not deemed a real ITU group, and was consequently not given a time slot within the ITU tower at Geneva. Again, formal procedures had a high standing in the ITU.

The Team of Specialists that was drummed together in Bern, in early 1975, had considerable less experience and expertise than Niklaus Wirth. When I interviewed Remi Bourgonjon about his role in the group, he invoked the saying that "in the land of the blind the one-eyed man is king" when asked about his own background and his role in the Team of Specialists.[18] In 1975, Bourgonjon was a young programmer employed at Philips Telecommunications at Hilversum in the Netherlands. He had never previously thought about designing a programming language. For two years, he had worked as a programmer assigned to a small computerised exchange project – a project that was brought to an abrupt end in 1974. As one of the first employees with a background in the emerging field of computer science, Bourgonjon had lobbied for the use of a high-level language within

[17] Kristen Rekdal, "Reiserapport fra møte i CCITT arbeidsgruppe XI/3 om Programmeringsspråk for datamaskinstyrte telefonsentraler", Geneve, 10 – 19 february 1975", 6 March 1975, box "CCITT - HLL Team of Specialists - Møtereferater 1974 - 1976, Møte 1-6", KRC. My translation.

[18] Remi Bourgonjon, interview with author, 16 January 2009, Heeze, the Netherlands.

Philips, which was still relying on assembler code in their software production. This lobbying did not immediately change the practice at Hilversum, but led Bourgonjon to a rather unsuspected career change: Almost by accident, he found himself given the responsibility of leading CCITT's Team of Specialists, even though his superiors were sceptical about its economic implications. According to Bourgonjon, Philips was, nevertheless, in a position where the CCITT activities could pay off, since the adoption of a high-level programming language was very much in the air at that moment, and Philips had come late to the party. If the CCITT standard would lead the telecommunication administrations to make this specific programming language a requirement, Philips wanted to have a head start. They had nothing to lose, since they were still programming in assembly. Disregarding the scepticism within Philips, Bourgonjon was given the opportunity to work on the CCITT assignment almost full time for the coming years. Before his appointment as the convenor of the Team of Specialists, Bourgonjon had participated in some of the early preparatory meetings in the CCITT. According to Bourgonjon, this participation was the reason behind his choice as group leader. Over the course of these meetings, Bourgonjon had raised his voice over what he considered to be rather uninformed comments made by other group members. These comments made people take notice, and when Wirth declined, the CCITT thought their next best option was Bourgonjon. According to Bourgonjon, he was both honoured and shocked at the same time, considering his age and experience.[19]

Figure 3.1 below is a picture of the core group, meeting in London in 1977 at the end of the study period, showing eight of the most active specialists working on the programming language. What is evident from this picture is, among other things, the age difference between the core members. Bourgonjon was really the youngest of all the team members, at 29, while the administration representatives, like Ken Clements of the UK Post Office, was a veteran when it came to international standardisation work and in his 50s.

[19] Remi Bourgonjon, telephone interview with author, 17 March 2011.


**Figure 3.1 The Team of Specialists.**[20]

Not all the participants had the same possibilities as Bourgonjon – some participated in the expert group part time, while others had a considerably narrower mandate. Furthermore, the participants were not equal in terms of expertise. This unevenness, in terms of time and expertise, contributed substantially to a design process that was fraught with difficulties. In addition, the fact that none of the key members of the group had English as a mother tongue and mastered the language to varying degrees contributed substantially to this unevenness. According to Bourgonjon, the first period, lasting for the whole of 1975 and well into 1976, was essentially "a period of chaos".[21] Conflicts of interest, as well as the difference in knowledge as well as communication skills, all contributed to this. The task of developing a programming language within an international team was an uphill battle, especially within a tight timeframe. Bourgonjon summarised this in a note to the team members:

[20] Back row: Jack Beierle (ITT), Heiko Sorgenfrei (Siemens), Hans Kvarneby (L M Ericsson), Trevor Boyd (UK Post Office), Katsumi Maruyama (NTT). Front row: Remi Bourgonjon (Philips), Ken Clements, (UK Post Office), Kristen Rekdal (RUNIT).

[21] Remi Bourgonjon, interview with author, 16 January 2009, Heeze, the Netherlands.

> […] this task is far from trivial. Experiences with comparable projects have shown that either, with an international team not working directly together, it will cost rather much time, or else the work must be done in a small group, working closely together. We are facing the problem of developing a programming language in an international team, not working very closely together, in a reasonable time.[22]

The anatomy and specific problems of this team are the subject of the coming pages, where I first sketch the structure of the cooperation, and then look into a few key issues in the technical design of the programming language.

## Coordinating contestations

From 26 January 1976 seven programming specialists met for ten days at the Swiss telecommunication administration headquarters in Bern to discuss the development of the standardised programming language for telephone switches. During the January meeting, tensions between the group members rose, as they were unable to agree on almost anything about the new programming language. The Norwegian researcher Svein Hallsteinsen participated and reported on the tensions to his supervisors:

> The impression after this meeting is that it will be impossible to agree on *one* language, unless it really consists of two languages. This is unacceptable to most of the participants. It is more likely that we will end up with two proposals, one European and one Japanese (PL/1 like).[23]

Hallsteinsen believed that the project could turn into what the CCITT's director Raymund Croze had called "hybrid monsters" at the general assembly of the organisation the year before, a bloated amalgam of every programming language that existed.[24] If not, the CCITT would have to vote on two different proposals, one coming from a broad group of European delegates and another one from the Japanese delegates that represented the NTT, the Japanese telecommunication administration. The sharp disagreement that Hallsteinsen had observed, between two wings within the

---

[22] Remi Bourgonjon to all members of the Team of Specialists, 6 March 1975, box "CCITT arbeidsdokumenter", KRC.

[23] The quote is from Svein Hallsteinsen, "Reiserapport fra møte i CCITT's spesialistgruppe for høynivå programmeringsspråk, Bern, 26. januar – 5. februar", 13 February 1976, box "CCITT - HLL Team of Specialists - Møtereferater 1974 - 1976, Møte 1-6", KRC. My translation.

[24] Minutes of the Plenary Meetings, CCITT Plenary Assembly 6 Orange Book Vol. I - IV 1976, CCITT, ITUA. 23. See chapter three for more about Raymond Croze's concerns about the development of technical standards within the CCITT.

expert group, was about features of the programming language. This tension between Japanese and European participants existed from the first meeting of the Team of Specialists, a tension felt by many of the participants throughout the project.[25] The number of documents contributed by the participating organisations illustrates these blocks, as the NTT, Philips and Runit were the main contributors.

| ***Organisation*** | ***Meeting*** | | | | | | | ***Total*** |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| NTT | 1 | 6 | 3 | 1 | 5 | 7 | - | 23 |
| UKPO | 1 | 3 | - | - | - | - | - | 4 |
| Swiss PTT | 1 | - | 2 | - | - | - | - | 3 |
| Philips | 2 | 4 | 6 | - | 4 | 3 | - | 19 |
| ITT | - | 1 | - | 1 | - | - | - | 2 |
| Siemens | - | 2 | - | - | - | 2 | - | 4 |
| LME | - | - | 1 | - | - | - | - | 1 |
| Dutch PTT | - | - | 1 | - | - | - | - | 1 |
| Runit | 2 | 2 | 4 | 3 | 6 | 3 | - | 20 |
| Runit/LME/Philips | - | - | - | - | 1 | 1 | - | 2 |
| The Team of Specialists | - | - | - | - | - | 1 | - | 1 |

**Table 3.3: Number of working documents.** [26]

Where the European participants, in general, could agree on some main features derived from existing programming languages similar to Algol, the Japanese would often come up with arguments that ran counter to the perceived wisdom among the other participants. Their favourite among existing programming languages was IBM's effort to create an ultimate high-level programming language for almost all purposes, the PL/1.[27] This message was reiterated time and time again by the main participant from the NTT, Katsumi Maruyama.

Both Algol and PL/1 were significant markers of different professional identities. Algol was, as discussed in chapter two, an important output of computer science, by some considered a paradigmatic exemplar of

[25] This is based on interviews with some of the participants, reviewing reports from the Norwegian delegates and official reports from the meetings.

[26] The documents are listed in COM-XI No. 19-E, Annex A, box Period 1977 – 1980, CCITT, ITUA.

[27] PL/1 held some technical similarities to Algol. On the history of PL/1, see George Radin, "The early history and characteristics of PL/I", *ACM SIGPLAN Notices* 13, no. 8 (1978); Sammet, *Programming languages: history and fundamentals*, 540-82.

the scientific approach to programming language design.[28] PL/1 had been created by the IBM and its user groups, called SHARE, in the mid-1960s. It was considered a multipurpose programming language that should cater for all uses, including systems programming and real-time systems, but in many ways it was considered an optimal substitute for Fortran.[29] Many technical similarities existed between the two, but the governance models that underpinned Algol and PL/1 were considerably different. Algol was in many ways a commons, a programming language governed by a self-appointed and self-regulating community of scientists, while PL/1 was the property of the IBM. However, there were also technical differences. One example of the schism came to the surface at the fourth meeting of the Team of Specialists, and was related to technicalities. The Norwegian researcher, Svein Hallsteinsen, commented on it in the following manner:

> It is a sharp disagreement within the Team of Specialists about whether security is an important property for a programming language for telephone switches. The PL/1 supporters consider the consistent use of variables a responsibility of the individual programmer and that security checks on compilation would severely reduce the flexibility of the language and consequently reduce programmer's productivity.[30]

The Japanese representatives wanted a less strict, and apparently less secure, solution, while the Europeans wanted something stricter – where the programmers' hands were tied considerably tighter than in the Japanese case. This was part of a wrangling over professional identity as well as strategic efforts to maintain the compatibility of already existing programming languages, where the European and the Japanese representatives held widely differing views and agendas. The main European representatives were steeped in mathematical computer science and its development virtues while the Japanese representatives were less concerned about such goals.
While the NTT was arguing for a PL/1-like language, Remi Bourgonjon from Philips and the Nordic telecommunication administrations wanted the

---

[28] Priestley, "Logic and the development of programming languages, 1930 - 1975".
[29] See Sammet, *Programming languages: history and fundamentals*, 547. On the history of SHARE, see Atsushi Akera, "Voluntarism and the Fruits of Collaboration", *Technology and Culture* 42, no. 4 (2001).
[30] Svein Hallsteinsen, "Reiserapport fra møte i CCITT's spesialistgruppe for høynivå programmeringsspråk, Bern, 26. januar – 5. februar", 13 February 1976, box "CCITT - HLL Team of Specialists - Møtereferater 1974 - 1976, Møte 1-6", KRC. This view is supported by the official document by the Sub-working party XI/3-2, "Progress report of the HLL Team of Specialists Fourth meeting: 26 January – 5 February 1976", Temporary document No. 29-E, box "CCITT - HLL Team of Specialists - Møtereferater 1974 - 1976, Møte 1-6", KRC.

standard to become an Algol-inspired language, which implied a more theoretical and mathematical foundation.[31] This was of no great surprise, as Algol was a dominating force in European computer science at the time, although generally shunned by the European computer industry. Furthermore, Bourgonjon was backed by the Belgium research laboratory of Philips, called the MBLE, which was considered to contain some of the best Algol specialists around.[32] The very first proposals stemming from Philips were written by Georges Louis, Paul Branquart and Paul Wodon of the MBLE.[33] The cooperation between Bourgonjon and the Belgian laboratory continued throughout the work on Chill, and according to Bourgonjon, the close cooperation with the researchers was a great inspiration and resource for him.[34]

The researchers hired by the Nordic telecommunication administrations, based at the research group at the computing centre of the Norwegian Institute of Technology in Trondheim (Runit) had a background in designing the programming language Mary, some sort of an Algol derivate.[35] The local team engaged in the CCITT work, people like Svein Hallsteinsen and Kristen Rekdal, had all been involved to some degree in the work on Mary. This had a lasting influence on the documents submitted by the Runit group, especially in the early period of the committee work, which often would use examples and experience gained through the Mary project as a point of departure.[36] The result was a series of small skirmishes between the Norway-based group and the Philips group. One example is the following quote:

---

[31] Peter, "The European side of the last phase of the development of ALGOL 60"; Alan, "The American side of the development of Algol".

[32] See, for example, P. Branquart et al., "The composition of semantics in Algol 68", *Commun. ACM* 14, no. 11 (1971).

[33] P. Branquart, G. Louis, P. Wodon, “Segments, a means for specifying access authorization”, Document A4, April 1975, box “Arbeidsdokumenter”, KRC.

[34] Remi Bourgonjon, interview with author, 16 January 2009, Heeze, the Netherlands.

[35] Rain, "Some formal language aspects of Mary or Algol X revistited".

[36] One example is Svein Hallsteinsen, Kristen Rekdal, Per Holager, ”Macros for CCITT HLL”, 4 December 1975, Document C2, box “CCITT Arbeidsdokumenter”, KRC.

> A parameterized text substitution mechanism like the one in MARY offers a reasonable implementation for several highly recommendable language features. However, it is a very flexible tool that requires some discipline on the part of the programmer. The alternative is an assortment of special constructs as proposed in (1), which will be safe and structured in use, but harder to learn and to implement. [37]

The latter alternative (denoted as 1) was proposed by Remi Bourgonjon. Here, it was presented in a discussion about how much flexibility the programmers would get in terms of so-called macros when using the future programming language, something mirroring some of the arguments used in the European versus Japanese discussions mentioned above. In some respects, the Runit proposal, which was largely inspired by the experience with Mary, and the general "machine-oriented higher-level languages" field in computer science, was more oriented towards efficiency of the compiled code rather than security. Again, the very design of the programming language was tied in with how the future user was conceived by the various parties of the quarrel.

When looking back at this debacle, Hallsteinsen, Bourgonjon and Rekdal all reinforced the impression of a particular tense relationship between the European participants and the Japanese one.[38] What about the Japanese themselves: did they share this impression? I have only been able to get in contact with Norio Sato, who served in the CCITT working group from around 1981.[39] The two Japanese participants in the original Team of Specialists, K. Maruyama and T. Koizumi, both attended some, but not all, of the meetings held in the team. I have not been able to interview them. However, the impression I got from Sato was that none of the Japanese participants was altogether happy with the results of the international cooperation.[40]

The tensions influenced the behaviour of the participants. In an interview with Rekdal conducted almost 30 years after the start of the Chill project, he commented on the process in the following way: "Of course, you never talk politics in groups like these. Everything is formulated in technical terms. Every problem is converted into technical problems, so they sound factual. It was a big challenge for an engineer, to manage the political

[37] Ibid., 13 – 14. (1) is a reference to R. H. Bourgonjon, "Macros and High level language", August 1975, Document B9, box "CCITT Arbeidsdokumenter", KRC.
[38] Interviews with Svein Hallsteinsen, Remi Bourgonjon and Kristen Rekdal. See the list of interviews at the back for details.
[39] Norio Sato, e-mails to the author, November 2008.
[40] Ibid.

problems in all this."[41] This process of translating everything into technical lingo found favourable conditions in the geographically widespread team organisation. Similarly, Bourgonjon underscored that forming alliances through mutual contributions in the form of written documents would ease the tensions, something that appeared for the very first time at the sixth meeting of the Team of Specialists in a document cooperatively prepared by Hallsteinsen, Rekdal, Bourgonjon and Kvarneby.[42] This came into being after several informal meetings of members of the Team of Specialists.

Apparently, this strategy worked well. In Kristen Rekdal's report from the fifth meeting of the Team of Specialists, the tone far more positive than the earlier gloomy reports. "This is one of the best meetings we have had. The Japanese attitude is considerably more flexible and ITT has declared support for the work. For the first time, we also got through the full agenda."[43] These positive signs marked the ending of the "period of chaos" and also implied that the framework for the end product, Chill, was emerging through the work of a real manual for the programming language. This change was based on an effort to orchestrate agreements, or to coordinate congestions, in particular between the two groups that worked full time on the project, at Philips and Runit.

## Maintaining compatibility

Professional identities aside, the wrangling over which existing programming language should provide the blueprint for the future CCITT programming language influenced the strategic manoeuvring of the large industrial organisations that participated through their delegates. Maintaining compatibility between the future proposal and an already existing programming language could greatly reduce the cost of compliance for any of the firms involved.

In one of the very first documents submitted by the NTT, which was actually a full proposal for a programming language, filed at the second meeting of the Team of Specialists in August and September 1975, these priorities were evident. The aim was to maintain compatibility with their own DPL language and PL/1, as well as a "basic feature integration of Pascal, ESPL-1, Pape, DPL, Mary, PLEX and TPL-2, taking standard

[41] Kristen Rekdal, interview by Lars Thue and Gard Paulsen, 13 September 2004, Trondheim, Norway. My translation.
[42] R. H. Bourgonjon, S. Hallsteinsen, H. Kvarneby, K. Rekdal, "Some proposed modifications to the revised manual", Document F2, box "CCITT Arbeidsdokumenter II", KRC.
[43] Kristen Rekdal, "Reiserapport fra 5. Møte i CCITTs spesialistgruppe for høynivå programmeringsspråk, Bern 10. – 21. mai 1976," 22 July 1976, box "CCITT-HLL Team of Specialists Møterapporter mote 1-6", KRC. My translation.

electronic switching systems into considerations".[44] This proposed amalgam language was based on a PL/1 notation, but added a few unique features found in all the seven shortlisted languages. In essence, though, it was 80% DPL, the proposed and specialised language of the NTT, hence the dual lopsidedness.[45]

The NTT's initial efforts can be understood as a defence of their own position and their own programming language, DPL, rather than a full-on support for PL/1, even though they believed in the general purpose programming language from IBM. According to Kristen Rekdal, the Japanese participants would repeatedly invoke the following line of arguments: since Algol 60 and 68 were developed mainly within academia, Algol was therefore perceived as an *academic* language; PL/1, on the other hand, was developed by IBM, and was therefore an *industrial* language. Since the new CCITT language was to be used for developing industrial products, it had to be an industrial language, and therefore it had to be PL/1-like.[46] To the Japanese, the PL/1 language implied efficiency, both in terms of learning and execution. However, this line of argument did not convince everyone. For one thing, it did not consider the fact that PL/1 had some distinct Algol flavours thrown into its mix.

Is there any evidence, in writing, that the other participants defended their own programming language in a similar way? L. M. Ericsson was represented in the group by Hans Kvarneby. L. M. Ericsson's programming language Plex had, just like the DPL-language of the NTT, been considered a viable candidate by the CCITT in the review round. Nevertheless, Kvarneby submitted only one sole-authored working document, "On separately compiled modules", in 1975.[47] This document cannot be interpreted as an effort to shift the standardisation effort in the direction of Plex, as it is more or less a paper that supports a technical solution proposed by Remi Bourgonjon and Philips.[48]

The ITT, which had their ESPL/1 language favourably reviewed in the initial assessment, submitted one document in 1975 and another in 1976.[49] The 1975 paper was only two pages long, and did not contribute or argue in any particular direction. The paper cannot in any way be understood as an

---

[44] NTT, "A proposal of a programming language for electronic switching systems", Document B18, box "CCITT Arbeidsdokumenter", KRC.
[45] "Remarks on the language proposed by N.T.T.", Temporary Document No. 201-E, box "CCITT Arbeidsdokumenter", KRC.
[46] Kristen Rekdal, interview with author, 18 June 2008, Oslo, Norway.
[47] Hans Kvarneby, L. M. Ericsson, "On separately Compiled Modules", Document C10, box "CCITT Arbeidsdokumenter", KRC.
[48] Ibid.
[49] R.J. Parente, "Comments on separately Compilable Modules and debugging tools", Document B12, box "CCITT Arbeidsdokumenter", KRC.

argument for particular functions found in their own programming language ESPL-1. The second paper submitted by ITT was called "Use of Based Variables in Systems written in ESPL-1."[50] As evident from the title of the paper, this hand-written and rather short document was based on the use of ITT's programming language ESPL-1. However, it reads more as a comment on a very particular technical question, rather than a solid proposition for how this question should be tackled in the new programming language.

The fourth manufacturer represented in the group, Siemens, did not have a programming language reviewed by the CCITT in 1974. They submitted only two working documents in 1975, "Proposed flow of control statements" and "I/O functions in a HLL for electronic switching".[51] Neither can be understood as an attempt to swing the development effort in any particular direction, as it reads more as an investigation of the seven shortlisted languages and contains only very general propositions.

As pointed out earlier, Kristen Rekdal acted on behalf of the research establishment Runit and the Nordic telecommunication administrations. The origin of this cooperation was the early proposal of Mary as a contender language in the CCITT review process. As such, the actions of Rekdal and Runit could be understood as a similar compatibility-seeking manoeuvre like those discussed above, although the vested interests in Mary were considerably smaller compared with those of the manufacturers that had applied in-house programming languages in existing products. Nevertheless, Mary had a lasting influence on the documents submitted by the Runit group, especially in the early period of the committee work, which often would use examples and experience gained through the Mary project as a point of departure.[52]

## Orchestrating agreements

In 1976 agreements replaced tensions, quarrels and disputes on several levels. The Team of Specialists became more unified and finally agreed on a proposal for a CCITT recommendation, which was filed as "the blue

[50] D. Copen, January 23 1976, "Use of Based variables in Systems written in ESPL-1. Some examples", Document D4, box "Arbeidsdokumenter II", KRC.

[51] Siemens Aktiengesellschaft, "Proposed Flow-of-Control Statements for the CCITT HLL", 7 august 1975, and "I/O functions in a HLL for electronic switching systems", 7 august 1975, Document B13 and B14, both in box "CCITT Arbeidsdokumenter", KRC.

[52] One example is Svein Hallsteinsen, Kristen Rekdal, Per Holager, "Macros for CCITT HLL", 4 December 1975, Document C2, box "CCITT Arbeidsdokumenter", KRC.

document" in May 1977.[53] This document compiled all the different ideas about what a programming language should be into one rough draft. The document was approved by the formal CCITT hierarchy, which agreed to prolong the programming language project until 1980. This ensured that the language would be developed further from the rough draft of the "blue document" to a more capable programming language three years later. During and between the fourth, fifth and sixth meetings of the Team of Specialists, from late 1975 to autumn 1976, a coherent framework was agreed and supported by most of the group's members. The "period of chaos" was replaced by a year of agreements. This was a period of practical work, as the team had to decide how they should describe, formulate and define the programming language they had argued about for the last year, in essence an exercise in writing down everything about the language, such as its syntax and its intended semantics. The practical work on the manual forced the participants into a more concrete way of working. This finally turned this awkward squad into something of a top-level troop, more effectively dealing with their differences than ever before. However, a real platoon it was not.

Still, disputes could escalate quickly. As they more or less followed a common pattern, this made them easier to avoid. Some of the most intense debates and quarrels could be postponed, while others were bypassed by making compromises: some of these compromises were even in the direction of the dreaded possibility of being PL/1-like, proposed by the ever more unlikely group of Europeans.[54] Since the two only participants that worked more or less full time on this project were Remi Bourgonjon and Kristen Rekdal, the majority of the manual writing was done in Hilversum and Trondheim.

On which direction did they agree? The committee tried, as they worked their way through each and every language concept, to strike a balance between the need for efficiency and flexibility on the one side, and reliability on the other – inset with the overall goal of achieving machine (and, inherently, manufacturer) independence and program portability. At an overall level, this balance was sought by aligning the background in computer science that many of the team members shared with the virtues of the community of electrical engineers that dominated the CCITT and the

[53] The HLL Team of Specialists, "Proposal for a recommendation for a C.C.I.T.T High-level programming language", The Blue Document, May 1977. The document is available at the ITUA.

[54] Kristen Rekdal, Svein Hallsteinsen, Per Holager, "CCITT HLL and Based Variables", 22 January 1976, Document D2, box "CCITT Arbeidsdokumenter II", KRC. This note explored the possibilities of fitting the based variable concept of PL/1 into the CCITT HLL.

telecommunications world in general. By arguing that certain features of the programming language could contribute to the production of more reliable software, the programming language would at least appear to be very much in line with the reliability focus with which the telecommunication world was preoccupied.

The group would agree on specific language features that were believed to be related to high reliability, such as the possibility to check as much as possible at the time of compilation, or in other words, at the moment the programming code was translated into machine code. Efficiency was also deemed to be one of the most important requirements at a time when computing power was very much a limited resource. The obvious way to achieve this was to bypass the high-level constructs and dip into machine code. Mechanisms that allowed for such toe dipping were consequently a major discussion point at the team meetings. It would most certainly contradict the philosophy of a high-level language in the first place – and make it less portable. More importantly, extensive use of low-level features was believed to degrade the program reliability. Too many programmers would bypass the programming language and design fast but erroneous low-level hacks. To marry efficiency and reliability was impossible. Consequently, the language had to strike a fine balance and reach a number of compromises. However, to be able to reach a number of agreements, some of the most debatable topics were postponed, in particular those related to low-level constructs.

The end result was a manual that described the vocabulary of the proposed language, including a formal syntax and informal semantics. This was not, however, something that came by itself. The first set of drafts of the manual was "inconsistent both in syntactic and semantic description".[55] The inconsistency was seen as highly unsatisfactory, not only from a scientific point of view, but also because the goal of making this language a standard. As Bourgonjon had stated, "for standardisation reasons, the descriptions should be unambiguous and complete". The real question then, is how could the group create consistency when progress by compromise was the 'plate de jour'?

## Descriptions and alignments

During the summer of 1976 the first substantive draft manual was finished. The inconsistencies were gradually ironed out in meetings throughout 1976

[55] Remi Bourgonjon to the members of the HLL Team of Specialists, Hilversum, 7 September 1976, Document F1, box "CCITT Arbeidsdokumenter II", KRC.

and one last editorial meeting in London in February 1977.[56] By applying what was described as an extended Backus-Naur Form syntax, the work was eased out by applying the state of the art in computer science concerning syntax description. The Backus-Naur Form was a metasyntax used to express context-free grammars: that is, a formal way to describe formal languages. It was created by John Backus as part of the creation of the programming language Algol in the late 1950s. The Danish computer scientist Peter Naur simplified Backus's initial design, which became known as the BNF during the 1960s.[57] Again, the team went along with ongoing concerns in the wider community of computer scientists to be able to achieve the goals set by themselves as well as the CCITT: a consistent programming language.

The moves leading towards a provisional and more consistent language description were also a result of a growing alignment of the different attitudes towards programming languages by the members of the Team of Specialists. However, these alignments also depended on local "support teams", in particular work within the MBLE in Belgium, which was the Belgian subsidiary of Philips, and at Runit in Trondheim. During 1976, the involvement with the MBLE would not only influence the thinking of Bourgonjon, but would also function as a meeting place between the Norwegian researchers involved in the Runit-led project and the Dutch researchers and language designers, as meetings between these actors were held at the MBLE.[58] Furthermore, similar meetings were arranged at L. M. Ericsson in Sweden in August 1976. These meetings that were held outside the team's official agenda would result in co-written contributions to that agenda, which dramatically increased their possibility of being approved by the others.

By the end of 1976, the Team of Specialists had agreed on the central features of the language and the "Crozian monster" envisioned by Svein Hallsteinsen was avoided. The next step would be to gather support for the language beyond the participating organisations. One particularly telling example was the last "official" meeting of the Team of Specialists, held in Kyoto, Japan, in October 1976. The meeting was devoted to the "syntax for the full CCITT HLL language proposal", and the participants were able to

[56] For reports on the meetings, see the official report: The HLL Team of Specialist, "Progress report of the HLL Team of Specialists", March 1977, COM XI No. 19-E, COM XI 1978 – 1980, ITUA.

[57] See the contribution of John Backus and Peter Naur in Wexelblat, *History of programming languages*.

[58] Kristen Rekdal, "Reiserapport fra uformelt møte om CCITT's høynivåspråk i Brüssel 6. 8. april 1976", 21 April 1976, box "CCITT-HLL Team of Specialists Møterapporter mote 1-6", KRC.

agree on how the syntax should be described in the language manual.[59] The meeting in Kyoto was held at the same venue as the large International Switching Symposium (ISS), a major trade meeting and technical conference held biannually at various locations throughout the world.[60] Just days after agreeing upon the syntax on their new programming language, the team could discuss it with other interested parties in the larger telecommunication world. Here, the group could gather support and align their proposal – or for that matter, they could also risk receiving a blatant rejection from the important decision makers.

Remi Bourgonjon presented the progress of the Team of Specialists in a session devoted to the concept of high-level languages in telecommunication systems, together with presentations from researchers from the NTT, the ITT and the French CNET laboratory.[61] Bourgonjon's paper made the influence from computer science explicit with references to publications by well-known figures from the computer science field, such as Niklaus Wirth, Per Brinch Hansen, Tony Hoare, David Parnas and Jean Ichbiah.[62]

That manufacturing firms like ITT, L. M. Ericsson, Bell and Siemens all presented various papers on aspects of the use and creation of software in telecommunication systems confirmed the importance that programming and software development steadily gained within the field of telecommunications. However, there were few other signs of formal backing of the CCITT initiative at the ISS. Only Bourgonjon could break the news that it was decided that Philips would use the programming language at the ISS.[63] The NTT, which had participated in the Team of Specialists, however uncooperatively, presented their continued work on the programming language DPL at ISS.[64] The NTT did, however, agree that they would modify DPL to a certain extent so that it would be in accordance with the

---

[59] HLL Team of Specialists, "Progress report of the sixth meeting of the HLL Team of Specialists, Kyoto, 12-22 October 1977", box "CCITT - HLL Team of Specialists - Møtereferater 1974 - 1976, Møte 1-6", KRC.

[60] "International Switching Symposium".

[61] Raymond Hubert Bourgonjon, "A High-level programming language for SPC Software Systems." Printed in Ibid.

[62] All names well known in the computer science field, although some of these would gain greater recognition a little later.

[63] This was reported by Kristen Rekdal. See Kristen Rekdal, "Reiserapport fra møte i CCITT's spesialistgruppe for høynivå programmeringsspråk, Kyoto, Japan 11. – 22 okt. 1976," 18 November 1976, box "CCITT - HLL Team of Specialists - Møtereferater 1974 - 1976, Møte 1-6", KRC.

[64] M. Kakuma, K. Maruyama, and T. Koizumi, "DPL-A High Level Programming Language for Electronic Switching Systems", in *International Switching Symposium* (Kyoto1976).

CCITT high-level language.[65] This underlined that a CCITT-approved programming language would enter a crowded "market".

The first public discussion of the work within the Team of Specialists was an indication of an agreement at the level of the technical community concerned with programming telecommunication systems. The 1976 ISS even coincided with the 100th anniversary of the invention of the telephone. In a celebratory article written by the chairman of the CCITT's study group XI (the parent group of the Team of Specialists), J. S. Ryan of the Bell Laboratory, acknowledged that "the programming of switching processors" had "become a separate art".[66] The Team of Specialists was perhaps not that interested in turning the programming of telecommunication into an "art," but separate it certainly was. Nevertheless, their work was duly noted as something of importance for the second century of the telephone. The endorsement from the study group leader, who presided over the work of the Team of Specialists in the CCITT, was an indication that the organisation looked on the Team's work with some satisfaction.

## Postponed deadlines and delayed concepts

The team had worked on the fringes of CCITT's hierarchical system for most of 1975 and 1976. The fringe status implied that they needed to gather support for their work within the ranks of the CCITT as they were closing in on their target. Since there was no way that a finished programming language could be expected before the deadline for proposals to the plenary meeting of the CCITT in 1977, the team was in need of a temporary seal of approval and a go ahead to further their work beyond this study period.

Long before the draft manuals were getting into any shape that would resemble anything useful, the Team of Specialists would have to report their results to the formal CCITT hierarchy. All the various parenting groups declared themselves "pleased at the progress made by the Team of Specialists", even though they had not come very far.[67] They also realised that a satisfactory proposal only could be fulfilled in the next plenary period. The Team of Specialists received an endorsement to continue, but only with some new and modified work assignments. One was that they would have to publish their preliminary results as a semi-official CCITT publication in 1977, which again forced the group to put their ideas down on paper sooner rather than later. In the end, the 1977 Plenary Assembly stated that work in

---

[65] Kristen Rekdal, "Reiserapport fra møte i CCITT's spesialistgruppe for høynivå programmeringsspråk, Kyoto, Japan 11. – 22 okt. 1976", 18 November 1976, box "CCITT - HLL Team of Specialists - Møtereferater 1974 - 1976, Møte 1-6", KRC.

[66] J. S. Ryan, "Signalling and Switching as we enter the second century", *Telecommunication Journal* 43, no. 3 (1973).

[67] COM XI-Temp. 2-E, box "Arbeidsdokumenter 2", KRC.

the next study period would be concentrated on three particular issues. Firstly, experience from evaluation and implementation had to be gained, which was also something that was proposed by the members of the Team of Specialists. Secondly, there had to be efforts to strengthen the relation between the three different languages created by the CCITT: the high-level programming language Chill, the specification and description language SDL and the man-machine language MML. Finally, the Plenary asked the working groups to consider "the aim of establishing a common standard terminology over the entire field of telephony and computer sciences ".[68]

The first point was obvious: before unleashing a programming language as a standard for the telecommunication world, it had to be used in a setting beyond small academic exercises conducted by a small number of players. The effect was the formation of the Implementors' Forum from 1977, which replaced the narrowly focused Team of Specialists. The two last points, however, stressed a tension that I have not considered in any substantial way previously: the tensions between different working groups within the CCITT hierarchy. While the last point only hints at a wish for greater linguistic coherence, the second one proved all the more serious, and one that I have not dealt with in any great detail previously: integration between Chill, SDL and MML.

The study group 11 of the CCITT and ultimately the CCITT's Plenary Assembly decided that the two other languages should be closely integrated with the work on the programming language, which in the period studied in this chapter was an almost non-existent consideration. The SDL language was an effort to standardise the graphical representations of the functionality of telecommunication systems. During the study period, the participants from L. M. Ericsson pushed the SDL group in the direction of a high-level description language, rather than standardised pictograms for advanced flow charts. This direction made the working group envisage a tighter integration of the programming language and the description language, or even a full integration. This triggered the idea of closely aligning the two languages – an idea that had not been discussed widely within the Team of Specialists. Such an alignment would mean a closer cooperation between the electrical engineers of the SDL group and the computer scientists dominating the Chill group.[69] Whether this move was part of a grand scheme by L. M. Ericsson, which would have to involve the inactivity in the Team of Specialists and their high ambitions of the work on SDL and the eventual proposal to integrate the two efforts, is not known.[70] Anyway, the call for integration

[68] COM XI-No. 1-E, 110, CCITT, Period 1977 – 1980, ITUA.
[69] Hemdal, "AXE 10 - Software Structure and Features".
[70] Later activities of L. M. Ericsson representatives in the subsequent study period do not indicate such a strategy. See chapter four for details.

was largely ignored by the two groups, partly because of the technical difficulties, but mostly because each group had enough to deal with already, and could not take on the extra load of further coordination.

The Team of Specialists had on purpose left out some important issues that needed decisions. Firstly, the bargaining process about whether and ultimately how the language should provide access to very low-level constructs was put off beyond the life of the group itself. Secondly, the issue of concurrency was also delayed, partly because of internal wrangling as well the state of the art. Concurrency, which basically implied the possibility of several operations executing and overlapping in time, and potentially interacting with each other, was at the time little understood or developed in the computer science field, and was perhaps best left for later.[71]

Consequently, when the so-called Blue Document was issued in May 1977, the group had come one step closer to creating a general purpose programming language, but created with a rather particular interest in mind, namely that of telephone switching systems. The group had argued that the range of applications used in a switching system was so broad that what was needed was a language with general capabilities. However, it was lacking in a number of areas. It was not integrated with other CCITT technologies that were developed in parallel, like SDL. It lacked concepts for handling concurrency in the language. In addition, several of the language constructs were inconsistent. All this was left to the group's successor, the Implementors' Forum.

[71] The first programming language that has been credited with incorporating concurrency at a fundamental level was Concurrent Pascal, created by the Danish/American computer scientist Per Brinch Hansen in 1974. See Per Brinch Hansen, "The invention of concurrent programming", in *The Origin of concurrent programming: From Semaphores to Remote Procedure Calls*, ed. Per Brinch Hansen (New York: Springer-Verlag, 2001).


**Figure 3.2 The CCITT Study Group XI meeting in 1976.**[72]

What did the participants think of the results? In the beginning of May 1977, Kristen Rekdal delivered his report on Runit's participation in the CCITT Team of Specialists to the Norwegian Telecommunication Administration. Here, he outlined his experiences and the results of the projects. He argued that the group was riddled with "contentions both between manufacturers and administrations, and between the individual manufacturers". [73] According to Rekdal, the administrations favoured standardisation in general, as it could ease the operation of their networks and perhaps reduce costs. The manufacturers were interested in better tools, but only if it put them ahead of the competition. Thus, standardisation for its own sake was of little interest, and was seen as bad if it costs anything. If one could arrive at some sort of compatibility between the programming languages already used by the manufacturers and the future recommendation, it would obviously be a fortunate position for a manufacturer. Another reason for the disagreements was the background of the team's members. Rekdal noted the

[72] Remi Bourgonjon, in the middle. The picture is reproduced with the permission of Remi Bourgonjon

[73] Kristen Rekdal, "Nordic participation in the development of The CCITT High Level Language – Final Report", 3 May 1977, Runit report STF14 A77016, box "Nordisk Chill-prosjekt", KRC.

following to his superiors:

> Both in cultural background, education and knowledge of English, the Team was inhomogeneous. Sometimes disagreements were caused by pure misunderstandings. In the beginning, this was a problem. In spite of these differences and many heated discussions, personal relations within the Team have been good. We are still friends and on talking terms.[74]

When the Team was finally dissolved after its final meeting in London in February 1977, it had survived a "period of chaos", to use the Bourgonjon's expression, and had escaped with a language description that merited further work. The consensus that the Team arrived at was a result of aligning the very different backgrounds of the team members, and the difficulty of gaining any strategic position within the Team by running solo. The alignments and the alliances were the key to the progress, as none of the efforts towards retaining compatibility between Chill and existing programming languages used in telecommunications was successful.

## The structure of collaboration

The account of the Team of Specialists has so far focused on the process of technical diplomacy. The route towards the interim halt of the programming language project that was the Blue Document could do with a second approach: one concerned with the structure of the negotiations, but one that can retain the processual outlook. A further investigation of who held prominent positions in the group and who was "in the thick of things" is pertinent.[75] In the following, I apply a social network analysis (SNA) to the task of teasing out a number of structural features of the work done in the Team of Specialists, which supplements as well as questions parts of the story told above.

The specific case of the structure of the Team of Specialists is of limited complexity, and a full-blown and formal SNA study might seem unnecessarily complicated when dealing with something that can be highlighted through simple frequency counts, such as the number of contributions made by different organisations to each meeting. However, a network analysis provides a more robust measurement of the work that went on within the Team, measurements that are possible to compare with the structure of the collaboration in later periods.

Only 11 participants were represented in the group, and there were never more than eight representatives at the same meeting. The number of

---

[74] Ibid.

[75] Linton C. Freeman, "Centrality in social networks conceptual clarification", *Social Networks* 1, no. 3 (1978).

official contributions, in form of written and submitted proposals and working documents, were about 70 distributed across seven meetings. I focus on two measurements. Firstly, the relationships formed through mutual participation in official meetings are measured in the joint appearance model. Secondly, the importance of contributions made by team members, or their parent organisation, to different meetings is measured in a model that estimates the participants' willingness to influence each meeting.

This makes it possible to distinguish between importance (or centrality, to use the lingo of SNA) gained through participation and an apparent centrality due to the willingness to exert influence through contributions to the various meetings. Both the joint appearance network and the influence model are based on measurements of the number of relations each individual has formed and the strength of these ties. However, there is an important difference: The joint appearance network has been projected onto a one-mode network where the tie strengths are measured by co-participation at each meeting during the study period of the Team of Specialists (meaning that the "heaviest" ties come from the most co-appearances). The influence network measures the strength of the ties as stemming from the number of contributed documents to each meeting by the parent organisation or the participant, joined with co-appearances. To be able to calculate the centrality indices, the original affiliation network has been projected onto a one-mode network, where the sum of the number of documents and the appearances is the weight of the different ties in the network, rather than just the multiplied co-appearance measure used before.[76] The crucial question when applying this measure is the relative importance of tie weights to the number of ties in weighted networks. In the following, I have viewed the tie weights as the most important, so that the ties with large weights are considered to have a greater impact than ties with only small weights.

[76] Contributions made by organisations that did not have any representatives at the meetings are, as a consequence, not included. I have also excluded documents credited to the whole Team of Specialists, while documents written by more than one organisation have been included in the count. This is because the document in question, the one credited to the Team of Specialists, was the full language proposal and not a document used in the discussion in the group.

| | Participants | Meetings | Ties | Density |
|---|---|---|---|---|
| ToS | 14 | 8 | 314 | 0.765 |

**Table 3.4: Descriptive statistics of the Team of Specialists measured as a network.**[77]

The high density, which measures the cohesion of the two-mode affiliation network and the clustering co-efficient, pays testimony to the tight integration that was evident in the Team of Specialists.

The centrality of nodes has always been the key issue in SNA.[78] Degree centrality has been a particularly important measure, being defined as the number of nodes to which a focal node is connected.[79] It is somewhat difficult to apply degree centrality to the influence-measuring model, as it was designed for binary networks and would disregard the weighted information. Although a number of attempts to apply this centrality measure to weighted networks, these attempts have focused on tie weights only, going too far in the opposite direction.[80] Even though I consider tie weights more important in both networks, disregarding the number of ties completely would be unfortunate. In an attempt to combine both degree and strength, I apply a measure recently defined by Tore Opsahl to ease the analysis of weighted networks.[81] This applies a tuning parameter to assess the relative importance of the number of ties compared to tie weights, and report a degree centrality measure, which is the product of the number of nodes to which a focal node is connected, and the average weight to these nodes adjusted by the tuning parameter. Following this, I report the normal degree centrality, and the variant of degree centrality that takes both the number of

[77] The density of the two mode networks is calculated in Ucinet. See Steve Borgatti, Martin Everett, and Lin Freeman, Ucinet 6 for Windows: Software for Social Network Analysis Ver. 6.0, Analytic Technologies, Harvard.

[78] Freeman, *The development of social network analysis: a study in the sociology of science*.

[79] Other measures exist. Typically, closeness and betweenness centrality are considered important. Closeness centrality is the inverse sum of shortest distances to all other nodes from a focal node, measuring how quickly a participant could reach others. Betweenness assesses how a node is able to channel the flow of a network as it calculates the degree to which a node lies on the shortest path between two other nodes. Both defined in ———, "Centrality in social networks conceptual clarification".

[80] Examples are listed in Tore Opsahl, Filip Agneessens, and John Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths", *Social Networks* 32, no. 3 (2010).

[81] Ibid.

ties and the tie weight into consideration.[82] Furthermore, a normalised degree centrality score is presented in the case of the joint appearance network, which is calculated from the raw binary affiliation matrix. This is primarily intended as a comparative measure, used more extensively in the next chapter.[83]

| Participant | *Joint appearance* Degree | Degree (Alpha) | Degree (norm) | *Willingness to influence* Degree | Degree (Alpha) |
|---|---|---|---|---|---|
| J. D. Beierle | 43 | 78.20 | 1.000 | 55 | 113.13 |
| R. Bourgonjon | 43 | 78.20 | 1.000 | 156 | 540.40 |
| R. T. Boyd | 7 | 7.00 | 0.143 | 21 | 34.02 |
| K. Clements | 14 | 17.46 | 0.286 | 15 | 18.37 |
| S. Hallsteinsen | 6 | 6.00 | 0.143 | 24 | 48.00 |
| H. Kvarneby | 35 | 65.48 | 0.857 | 47 | 101.89 |
| T. Koizumi | 13 | 15.62 | 0.286 | 20 | 29.81 |
| M. Kakuma | 8 | 8.00 | 0.143 | 8 | 8.00 |
| K. Maruyama | 37 | 64.97 | 0.857 | 75 | 187.50 |
| S. Ogawa | 8 | 8.00 | 0.143 | 8 | 8.00 |
| K. Rekdal | 37 | 64.97 | 0.857 | 137 | 462.90 |
| L. Sandberg | 17 | 24.78 | 0.429 | 27 | 49.60 |
| H. Sorgenfrei | 38 | 64.97 | 0.857 | 65 | 145.34 |
| T. Wakamoto | 8 | 8.00 | 0.143 | 8 | 8.00 |

**Table 3.5 Centrality measures of the Team of Specialists.**

The joint appearance network highlights the presence and ties between three important industrial participants, Bourgonjon of Philips, Sorgenfrei of Siemens and Beierle of the ITT. It also highlights less active participants from telecommunication administrations, except the appointed mutual representation of the Nordic administrations through the researcher Kristen Rekdal and the strong presence of Japanese participants representing the

[82] This follows Ibid. The joint measurement includes a tuning parameter, α, to control for the relative importance of the number of ties and the weight of the ties. The α parameter is set to 1.5, which weights tie weights as the most important.

[83] Normalised degree centrality on the co-appearance network normalises the scores against the maximum possible scores in an equivalently sized connected two-mode network and hence provides appropriately scaled measures. The score is calculated in Ucinet. See Borgatti, Everett, and Freeman, Ucinet 6 for Windows: Software for Social Network Analysis. Alpha-justified scores calculated with R and tnet. See Tore Opsahl, *Structure and Evolution of Weighted Networks* (University of London (Queen Mary College), London, UK, 2009); R Development Core Team, (Vienna, Austria: R Foundation for Statistical Computing, R: A Language and Environment for Statistical Computing).

NTT.[84] By considering the centrality indices, Beierle and Bourgonjon are ranked as the most central participants when considering co-appearances, but when taking the influence measurements into consideration, a somewhat different pattern emerges. The obvious lack of influence-gaining contributions from centrally positioned actors such as Sorgenfrei of Siemens and Bierle of the ITT render their ties in the network as weak, even though their appearance at all the meetings makes the number of ties (to meetings, this time) high. This also becomes clear when inspecting two comparable illustrations of the two networks, one based on the projected one-mode network of co-appearances, the other a joint display of meetings and participants based on the weighted influence network.[85] In the first illustration, the tie between participants is based on the number of co-appearances, while the weight of each tie in the latter illustration is calculated as a sum of participation and the number of written contributions made to each meeting rather than the total score, which is used in the centrality measures above. This makes it possible to get an impression of the variance of influence-seeking actions through time, and the intensity of document submission at various meetings coupled with who was present.

[84] On the setup of the telecommunication industry in Japan, see Fransman, *Japan's computer and communications industry : the evolution of industrial giants and global competitiveness*.

[85] Katherine Faust, "Using Correspondence Analysis for Joint Displays of Affiliation Networks", in *Models and Methods in Social Network Analysis*, ed. Peter J. Carrington, John Scott, and Stanley Wasserman (Cambridge: Cambridge University Press, 2005).

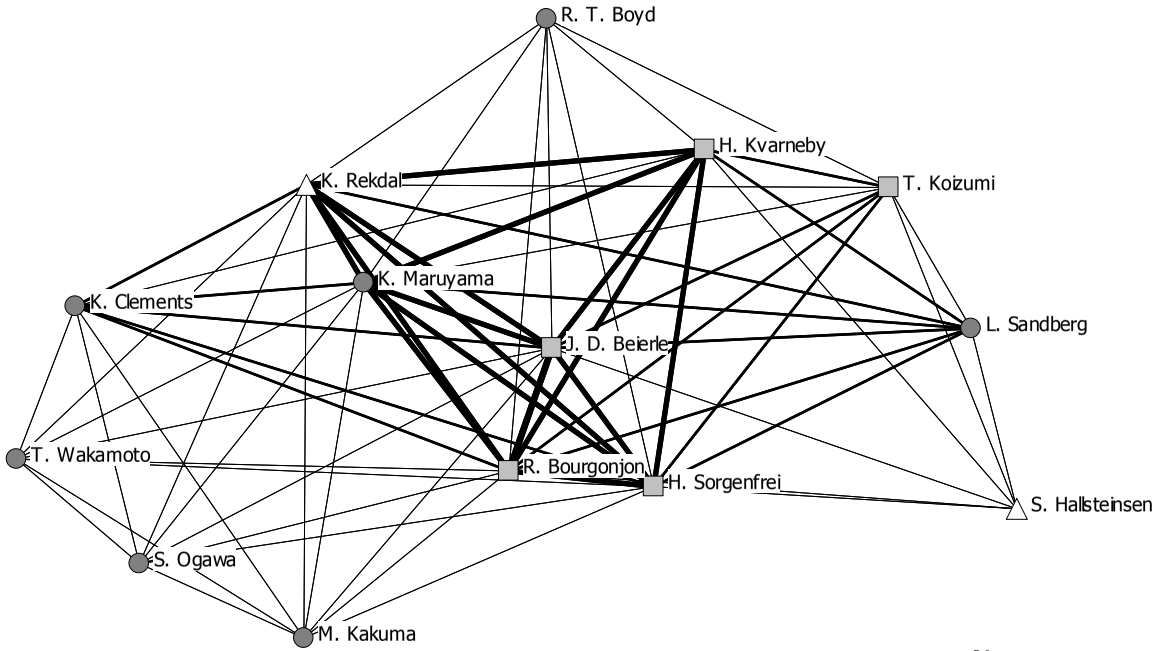


**Figure 3.3 Participants in the Team of Specialists, with ties weighted by co-appearance.**[86]

[86] The individuals listed in the figure participated together with the individuals to which they are linked. The width of the links illustrates the number of common meetings. Dark grey circles denote participants from telecommunication administrations, and white triangles are research institutions or other academic institutions. Light grey squares are from manufacturing firms. The position in the diagram illustrates the measured centrality of the participant, based on the spring embedding layout technique. Figure created using Netdraw. See Steven P. Borgatti, NetDraw: Graph Visualization Software. Ver. 2.097, Analytic Technologies, Harvard.

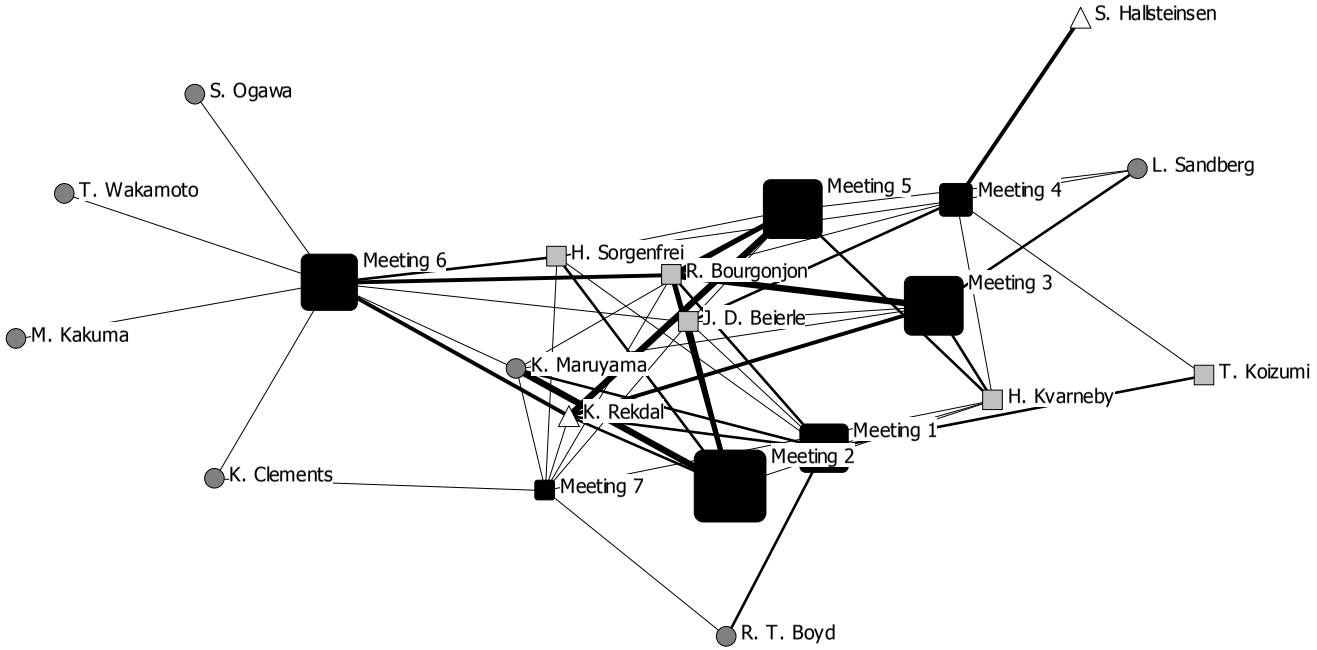


**Figure 3.4 The Team of Specialists, projected as affiliations to meetings with weighted ties.**[87]

[87] The width of the links illustrates the number of written contributions made by an individual who also participated at the meeting. The size of the squares illustrates contributions and participations at the meeting. Dark grey circles denote administration participants; white triangles are research institution participants and light grey squares are telecommunication administration delegates. The position in the diagram illustrates the measured centrality of the participant and the meeting, by the spring embedding technique. Figure created using Netdraw. See Ibid.

Summing up, based on analyses of joint appearances and influence ambitions, some structural features of the cooperation seem evident. Some participants had a strong presence at the group meetings but contributed little to the work in terms of written documents. This could be rooted in two different reasons. Either the actors participated in the work in order to observe what went on, but held no real interest in directing the work in any particular direction, or they held so little expertise that any meaningful contributions were impossible. I will consider each of these possibilities in the qualitative analysis below.

Both measurements have a number of other shortcomings. First of all, the simple metric of the number of contributions might indicate a willingness to influence, but it says little about whether this effort was successful or not. The number of contributions cannot be considered a part of some sort of out-numbering game. Second, the individual-oriented participation network does not take into the account the fact that some of the organisations sent different participants to different meetings, but retained continuity through local teams. Furthermore, it is important to stress that the SNA glances over important qualitative dimensions of the contributions, such as the quality of the argument, its consistency and so forth.

Co-affiliations illustrate the opportunities to influence the decision processes in the team, and the ability to form relationships through joint appearances made such influence all the more effective. The coupling of appearances and contributions reveals a number of actors who were less active than their position made possible. This is particularly so for the representatives of Siemens and the ITT, Sorgenfrei and Bierle. As such, the network analysis has revealed some additional features of the Team of Specialists that have not been evident in the archival material and interviews that formed the basis of the rest of the chapter. However, more importantly, it has strengthened the impression that the main players in the Team of Specialists were participants that shared a similar technical background in computer science, in particular Bourgonjon and Rekdal.

## Some conclusions

In this chapter I have described how the Team of Specialists traversed a number of hurdles towards a first sketch of a CCITT-recommended programming language. Professional differences and strategic manoeuvring were overcome through technical diplomacy, as a set of problems and difficulties were managed. This was more due to an alignment of the participants' ideas on programming language design than processes in the parent organisations of the participants. I have argued that many of these realignments were due to a shared understanding, a common development virtue if you like, that ran prior to the work in the Team of Specialists. However, a number of the finer ingredients in the diplomatic process were bound up in compromises made in a pragmatic fashion.

The first set of problems that faced the Team of Specialists was related to whether the future programming language should be an amalgam of existing languages used in telecommunications, or whether the group should try to build on the cutting edge of programming language design. Throughout the "period of chaos" much of the diplomatic negotiations stemmed from the strategic manoeuvring of administrations and manufacturers that had already invested in programming language technologies, wishing to regain compatibility with that and the future recommendation. Later on, this was replaced by more focused technical decision-making, which gradually bypassed the arguments for solutions grounded in the wider technical community. However, existing languages were not only technologies that held industrial ramifications, but were also markers of different professional identities. One of the most pronounced arguments within the team was the combined issue of the compatibility problem and the professional identity markers, as the struggle between the PL/1-supporting Japanese representatives and the Algol-oriented Europeans illustrated. Hurdle number three was related to the special needs in a telecommunication programming language, like how to handle parallelism and the relationship to SDL. In this chapter, I have highlighted the process and structure of the technical diplomacy that went on within the Team of Specialists, and how each of these three difficulties was managed

First of all, the period from 1975 to 1977 was about how a number of contestations were brought under control through a number of orchestrations: First, by aligning the work of Remi Bourgonjon at Philips and Kristen Rekdal at Runit, the Team of Specialists was able to produce a proposal for a recommendation on how a CCITT high-level programming language could look. Secondly, by engaging the full team in the practical work of writing a description and a manual for the language, the work became more practically oriented and consequently more consensus-based.

The structural analysis of the technical diplomacy in the Team of Specialists revealed additional aspects of the project. Important and visible participants in the Team, the actors habituating central positions in the network, were representatives of manufacturing firms.

The influence analysis revealed that a number of these central participants were, however, more observers than contributors. They were centrally positioned, but showed little willingness to influence.

This underlines the impression of the Team of Specialists as untypical of the CCITT. It was not dominated by representatives of administrations, even though they had been instrumental in its formation. The most effective participants were those who formed alliances based on a shared outlook on how programming should be done, and consequently, how a programming language should promote such virtuous programming. These alliances were, however, not only internal to the Team, but also pulled together external

participants, like Bourgonjon's contact with researchers in Philips' Belgian research laboratory MBLE and the support of the CCITT hierarchy.

Knowledge creation and knowledge adaptation at the crossroads of computer science and electrical engineering, within a bureaucratic organisation in transition, were never straightforward. Technical considerations on flexibility, security, efficiency and portability were all part of the design process. Some of them were postponed for later consideration, like the integration of the programming language and the description language (SDL), the creation of concepts allowing for concurrency and the availability of low-level constructs for hardware level access. All in all, during 1976 and early 1977, compromise and postponements led the project towards a partial fulfilment and the subsequent dissolution of the Team of Specialists.

During those two years, the Team of Specialists drew support for their work from outside resources, like non-committed firms and international organisations. In the summer of 1977, the Team of Specialists was to be dismantled. A new organisational entity, the so-called Implementers Forum, was about to embark on perhaps an even more difficult task than designing and agreeing on a high-level programming language: they were about to make it work in practice. This is the subject of the coming chapter.

# 4. Compromise and complexity: the implementation of a programming language

Starting in the summer of 1977 the programming language designed by the Team of Specialists was put through the hoops in extensive field trials. It was finally approved by the CCITT Plenary Assembly in November 1980 as an official recommendation.[1] During this period, new features were added to the language, existing ones were refined and the language was *implemented* through the construction of a set of compilers. Implementation and programming language design were combined and allowed to feed back into each other, as the participants in the CCITT made the language ready for programming duty. Throughout this period the language designers of Chill met with prospective users of the programming language in the Implementors' Forum, which was the continuation of the Team of Specialists.[2]

The Forum was characterised by collaboration, conflict, compromise and complexity. The participants in the Forum were able to collaborate on practical projects beyond the common language they were set to design, in particular on the construction of compilers. Still, there were heated debates and conflicts, in particular on unresolved programming language design issues that were bound up with conflicting development virtues and ideals. The way out of the first set of deadlocks was found in compromises, typically by aligning different actors through mutual contributions. However, some of the compromise solutions also added to the complexity of the programming language, duplicating features and constructs, so much that a leading computer scientist declared that Chill was "by far the most complex, uneven and compromise-ridden programming language the world has yet seen", when he first encountered it.[3] Such complexity added to the difficulty of implementing the language in any effective fashion, although several parties succeeded in developing compilers for Chill by the end of the period of the Implementors' Forum. Most implementors dealt with this duplication and complexity by implementing only those subsets of Chill that were suitable for the compilers' intended application.

---

[1] The language definition was published as CCITT *High Level Language (CHILL)*, CCITT Recommendation Z.200 (1980). On the ratification, see Minutes of the Plenary Meeting, 10 – 21, November 1980, ITUA.

[2] The progress reports from the Implementors' Forum are attached to COM XI-No. 270-E, Period 1977-1980, CCITT, ITUA.

[3] Dines Bjørner and Peter L. Haff, "A formal 'denotational' semantics definition of CHILL", Technical report ID888, September 14, 1979, ix, box "NTT 72-2/NT-P 1979 – 1980", KRC.

Despite the complexity, compromises and conflicts, the period also fostered a small community, and in the end, a language specification worthy of publication. What had been a small project partly embedded in a community of technological practitioners interested in programming telecommunication systems soon became a community on its own, the Chill community if you like. Considering the infighting, the tangles and the tensions that had marred the Team of Specialists and the Implementors' Forum, this was no small achievement. After the last meeting of the Implementors' Forum in Melbourne, in late September 1979, Rekdal noted they had "achieved what very few believed was possible five years ago", concluding with: "Chill is finished!"[4] The result was, he claimed, "a language with several novel features, and it is a very comprehensive language".[5]

This chapter tracks the last years of the Chill project before its ratification as an official recommendation by the CCITT in 1980.[6] It focuses on language design decisions, implementation issues, how the field trials were organised and how feedback was allowed into the language design process. I also analyse how the Chill community was developed and extended in the period from around the summer of 1977 until the programming language was published as a standard in 1980. Throughout the chapter, I investigate how community-level norms and organisational level strategies influenced the fate of Chill up until its final ratification.

## Design, implementation and feedbacks

Before we look into the activities of the Implementors' Forum in more detail, it can be useful to recapitulate the distinction between *language design, implementations* and *systems programming*. *Programming language design* is commonly understood as the efforts of putting together the definition of the lexical, syntactical and semantic elements of a language, hopefully fulfilling some sort of design criteria. The majority of the programming language design of Chill was done within the Team of Specialists. Here, the key features of the language, such as key words, concepts and capabilities, were decided. In particular, the design goals that

[4] Kristen Rekdal, Travel report, Melbourne 24 – 29 September 1976, CCITT – Implementors Forum SWP XI/3-2, WP/3 Melbourne, Sept. 1979, box "Implementors Forum 9. møte Melbourne, Sept. 1979, Serie O", KRC. My translation.

[5] Kristen Rekdal, "The Nordic CHILL Project", in *Runit Report* (Trondheim: Runit, 1980), 1.

[6] The account is again based on mainly two collection of sources, the ITU archive (ITUA) and the Kristen Rekdal Collection (KRC). Some documents not available in the KRC were complemented by the private collection of Remi Bourgonjon (hereby cited as RBC), which holds a complete set of official documents for the Implementors' Forum period.

made their way into the language already in the 1977 specification from the Team of Specialists were concerned with enhancing reliability by making the language constructs possible to check when compiling the code, placing severe restrictions on the intermixing of data types (so called strong typing, again to ensure reliability), to permit the generation of efficient machine code by adopting features from so-called machine-oriented higher level languages and still be flexible enough to cover a wide range of applications and different types of hardware.

However, the Forum would have to sort out some of the inconsistencies in the draft language sketched out by the Team of Specialists, as well as adding new features necessary for making the language viable for the telecommunications industry. As such, the Forum was an extended effort in language design. The ultimate result of the programming language design activities was the final language specification – which was published by the CCITT in 1980 as an official recommendation.[7] A language description is, however, nothing but text. To be able to use the programming language, one would need to *implement* the language, which was the main concern of the Forum.

*Programming language implementation* is most commonly understood as the creation of *compilers*. A compiler is a program that translates code written by programmers into code understandable to the targeted computers. As described in chapter two, computer hardware does not interpret code written in a programming language directly, as hardware deals with sequences of particular instructions in machine code. Therefore, the code written in a high-level programming language has to be translated into machine code before it can be processed by a computer. This translation can be automated and forms a program in itself. Such a translation program is called a *compiler*, and the text to be translated is called source code. Such a program is, in itself, characterised by three languages: its source language, being the high-level programming language used by the programmer, its object language, which is given by the target hardware platform, and the language in which it is written, being the implementation language.

In the 1950s and 1960s, compilers figured among the largest programming projects of the time, involving large investments and numerous man-years, and also attracted much research and development.[8] In the late 1970s, the situation was somewhat different. According to the important textbook on the subject, by Alfred V. Aho and Jeffrey D. Ullman from 1977, the effort needed to construct a compiler was much less than

[7] *CCITT High Level Language (CHILL)*, CCITT Recommendation Z.200 (1980).

[8] A reliable source on both the history and technical details of compilers is Niklaus Wirth, *Compiler construction*, International computer science series (Harlow, England ; Reading, Mass.: Addison-Wesley Pub. Co., 1996).

before: "[...] it is not unreasonable to expect a fairly substantial compiler to be implemented as a student project in a one-semester compiler design course." [9] One major reason for this was that the tasks involved in compilation had been thoroughly understood and that a shared understanding of the processes of compilation was available.

This typically involved a number of tasks that were common to any compiler, regardless of the source language or the object language. The process of compiling source code involved passing the code through several phases of lexical and syntactical analyses, optimisation and finally code generation. A figure illustrating the steps involved in the translation of code normally involved in a compiler as of the second half of the 1970s is produced below.

[9] Alfred V. Aho and Jeffrey D. Ullman, *Principles of compiler design*, Addison-Wesley series in computer science and information processing (Reading, Mass.: Addison-Wesley Pub. Co., 1977), 1.

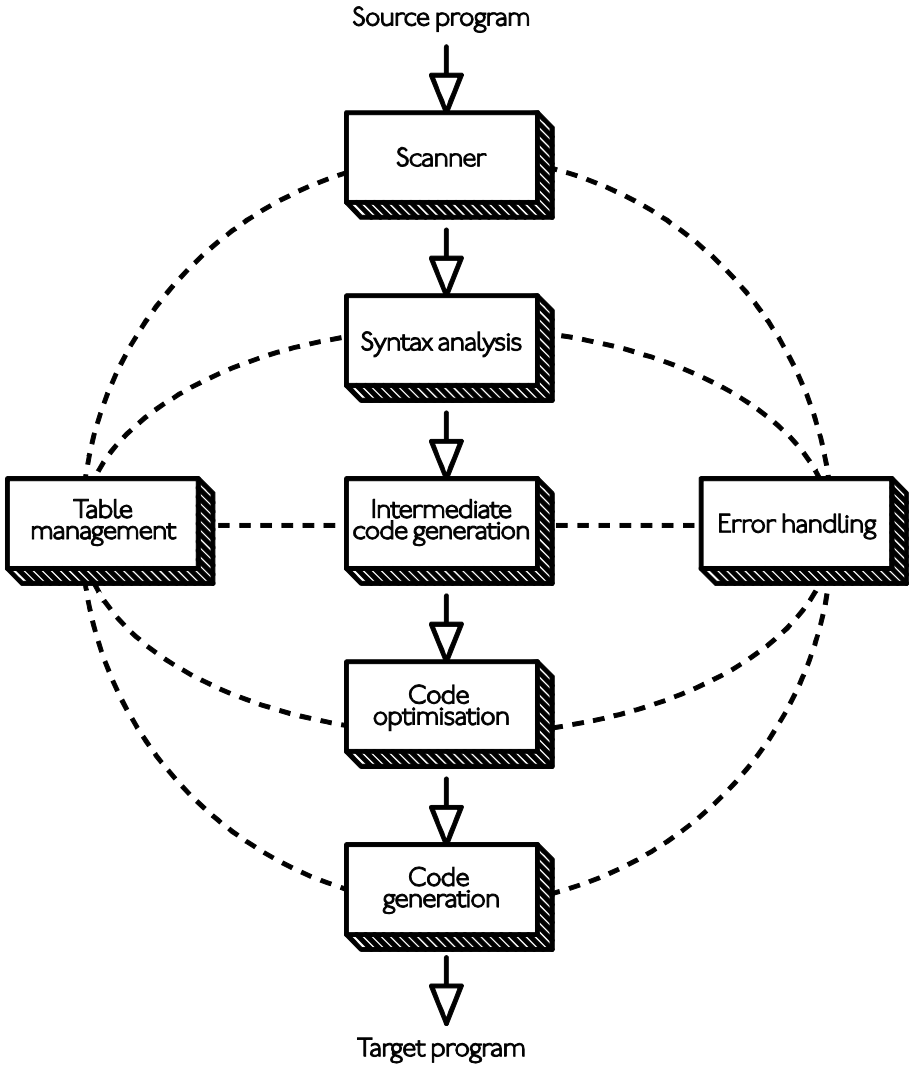


**Figure 4.1 Phases of compilation**

Regardless of the common aspects of all compiler design, to implement a programming language that was still in the making was a more complex undertaking, and more importantly, the quality of these early compilers had to be well beyond what was achievable for a single student in a one-semester course.

Kristen Rekdal, one of the most central participants in the Team of Specialists and the Implementors' Forum, described the importance of implementations and compiler construction in the following way:

> The practical usefulness of a programming language can only be demonstrated by using it for practical programming. Furthermore a language can only be evaluated together with its compiler. A good language can be rendered useless by a poor compiler, and a good compiler can compensate for language defects.[10]

In the end, however, the compiler is just a tool made for the programmers developing *systems* and *applications.* In programming, these two things are normally understood as quite separate undertakings. Application programming is the creation of software that provides services to a user while systems programming provides services to hardware. Considering Chill, it would be fair to assume the development of programs intended to solve a specific telecommunication problem, like a routing application or a billing procedure and so forth, were more similar to systems programming than applications programming, although the terminology was often mixed together. In this early phase, the availability of implementations was non-existent, so to start using the programming language an implementation project would often be a necessary precondition to take up actual systems or application programming projects.

Although the creation of a programming language inevitably moves along a path from design towards systems or application programming, it is not altogether linear. In reality, the boundaries between the periods were a lot less distinct and knowledge from implementation efforts and application programming efforts was allowed to feed back into revisions of the language design throughout its existence. The period of the Implementors' Forum makes this very clear, as the Forum would, at the same time, be active in compilation design and make numerous language revisions. Inevitably it was also a considerably larger group. While the members of the Team of Specialists had been restricted to a selected few, the Forum included a total of 70 different participants over the whole period.[11] The level of activity grew just as much, an indication being the 174 written contributions made to the Forum.[12] The Forum held nine lengthy meetings during a period of 24 months, clocking in 44 days of discussions, reviews and decision-making in various locations around the world. The 70 different delegates that participated in the Forum came from more than 25 different organisations,

[10] Rekdal, "The Nordic CHILL Project", 1.
[11] The number includes delegates but not observers.
[12] Contributed documents were listed in the nine official progress reports that the Implementors' Forum made to the official CCITT Working Party (XI/3-2). The total number includes all types of working documents.

spanning several telecommunication administrations, research organisations and industrial firms, and representing 16 different countries. In comparison to the period from 1975 to 1977, the Forum was certainly a lot more comprehensive and ambitious, both in terms of the number of participants and the activity level. The amount of work done on Chill-related projects in this period can be characterised as substantial and widespread. I have summarised some of the details about the Implementors' Forum in the table below.

| Meeting | Dates | Place | Participants | Documents |
|---|---|---|---|---|
| **1** | 12 – 16 September 1977 | London | 21 | 12 |
| **2** | 22 – 28 November 1977 | Geneva | 18 | 26 |
| **3** | 13 – 17 February 1978 | The Hague | 19 | 23 |
| **4** | 5 – 8 June 1978 | Geneva | 22 | 24 |
| **5** | 18 – 22 September 1978 | Tokyo | 18 | 12 |
| **6** | 12 – 15 December 1978 | London | 26 | 24 |
| **7** | 19 -22 February 1979 | Geneva | 36 | 18 |
| **8** | 14 – 18 May 1979 | Florence | 32 | 23 |
| **9** | 24 – 28 September 1979 | Melbourne | 31 | 12 |

**Table 4.1 Meetings in the Implementors' Forum, with number of participants and number of contributed documents.**[13]

The group was organised outside the formal CCITT hierarchy, but affiliated with the work of its parent Study Group within the CCITT. In parallel to the cooperation in the Implementors' Forum, several participants ran Chill-projects on the side, like the joint-Nordic compiler project and a cooperative effort between Philips and the Dutch administration, racking up the number of collaborations and the number of participants in Chill-related activities. In the following, I will look a bit more into the structure of the Implementors' Forum through the concepts of social network analysis, before I analyse how compilation construction and language design was intertwined in the period of the Forum.

## Implementing structure

The actors that had been active in the Team of Specialists dominated the Implementors' Forum. Remi Bourgonjon of Philips continued as the convenor of the group, with Kristen Rekdal of Runit working as the vice-

[13] Information from the official reports written by Remi Bourgonjon, most of which are available in KRC, except for reports on meetings number one and two, where the reports are only found in the RBC.

convenor. A number of manufacturing firms were present throughout the period, some new to the group, and others with experience from the Team of Specialists. The ITT, L. M. Ericsson, Western Electric, Siemens, Philips, Hasler, AT&T, as well as a couple of Japanese manufacturing firms were active participants.[14] Another group attending the meetings were researchers and scientists. One example was Dines Bjørner, a computer scientist from the Technical University of Denmark and acting as a specialist on behalf of the Danish telecommunication authority. Other participants with a similar background came from the Italian research organisation CSELT. Kristen Rekdal had held a similar position since the work started in the Team of Specialists, working on behalf of the Nordic telecommunication administrations, but based at the computer centre of the Norwegian Institute of Technology of Trondheim, Norway. The number of passive observers grew, including a number of sporadic attendees from the telecommunication administrations of Brazil, Hungary and the GDR, making up a fairly inconsistent and heterogeneous group. How was this group different to the Team of Specialists?

Following the social network analysis in chapter three, it is possible to review the structure of the Forum through the analysis of joint appearances and document contributions, as well as comparing some descriptive statistics of the Team of Specialists with the Implementors' Forum.

| | ***Participants*** | ***Meetings*** | ***Ties*** | ***Density*** |
|---|---|---|---|---|
| ToS | 14 | 8 | 314 | 0.765 |
| IF | 70 | 9 | 4636 | 0.319 |

**Table 4.2: A comparison of network measures of the Team of Specialists and the Implementors' Forum.[15]**

Whereas the Team was small and dense, in terms of frequencies as well as the density measures, the Forum was dispersed and much larger.[16] This structure is mirrored in the individual measures of centrality, which gives an indication of who were "in the thick of things", both in terms of a position where it could be possible to influence the network through affiliations and

[14] Study Group XI, Sub working party XI/3-2,"Progress report on the ninth meeting, Melbourne, 24 – 24 September 1979," Melbourne, 1-11 October 1979, Temporary Document No. 201, box "Implementors Forum 9. møte Melbourne, Sept. 1979, Serie O", KRC.

[15] The calculations follow the conventions established in chapter three.

[16] The density measure is highly sensitive of the size of the network. See Noah E. Friedkin, "The development of structure in random networks: an analysis of the effects of increasing network density on five measures of structure", *Social Networks* 3, no. 1 (1981).

in actual documented contributions.[17] A larger group of participants held high scores in terms of centrality in the Implementors' Forum, but relatively, the core group was smaller than in the Team of Specialists, due to the large group of participants attending only a few meetings of the Forum. In terms of organisational affiliations, the core members of the Forum, both in terms of participation and influence-seeking activities, were largely from manufacturing firms, with a few important exceptions. In particular, the representative of the Dutch telecommunication administration, R. W. Meijer, was an active member of the Forum.

The figures also highlight the general impression of the continuity between the Implementors' Forum and the participants in the Team of Specialists. The individuals listed in the figure participated in three meetings or more, together with the participants with which they are linked. We recognise the centrality of former Team of Specialists members Bourgonjon, Rekdal, Sorgenfrei and Clements. Some new individuals, such as the Italians Martucci and Benevolo, the Dutch telecommunication administration representative Meijer and the important participants Bjørner and Jacobson also figure in the social network analysis illustrations. The main group of new participants were European, coming from both manufacturers and administrations. In the Italian case, Benevolo represented the operator-owned research establishment CSELT while Martucci was from the manufacturing company SIT-Siemens, soon to be renamed Italtel.[18] Below, figures and tables that underline these general points are presented. The displays articulate the complexity of the Forum, while the tables make it possible to compare some of the centrality measures used in the study of the Team of Specialists. To recapitulate a bit, the measurements are focused on different ways of measuring degree centrality, which is the number of nodes to which a specific node is connected and indicate who were the most centrally positioned and active members of the network that was the Forum.[19] As discussed in the previous chapters at some length, all the measures have a number of deficiencies, although the set of general degree centrality, normalised scores and the tuned alpha-score were considered useful to the task at hand. Now, the normalised degree centrality scores of

[17] Freeman, "Centrality in social networks conceptual clarification".

[18] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 432.

[19] Other measures exist. Typically, closeness and betweenness centrality are considered important. Closeness centrality is the inverse sum of shortest distances to all other nodes from a focal node, measuring how quickly a participant could reach others. Betweenness assesses how a node is able to channel the flow of a network as it calculates the degree to which a node lies on the shortest path between two other nodes. Both defined in Freeman, "Centrality in social networks conceptual clarification".

the joint appearance network are particularly interesting, as they are more directly comparable to the numbers put forward in the previous chapter.[20] As discussed before, it is not viable to calculate a similar score on the willingness to influence scores, because of its inherent weighted nature.

[20] This follows Opsahl, Agneessens, and Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths". The joint measurement includes a tuning parameter, α, to control for the relative importance of the number of ties and the weight of the ties. The α parameter is set to 1.5, which weights tie weights as the most important.

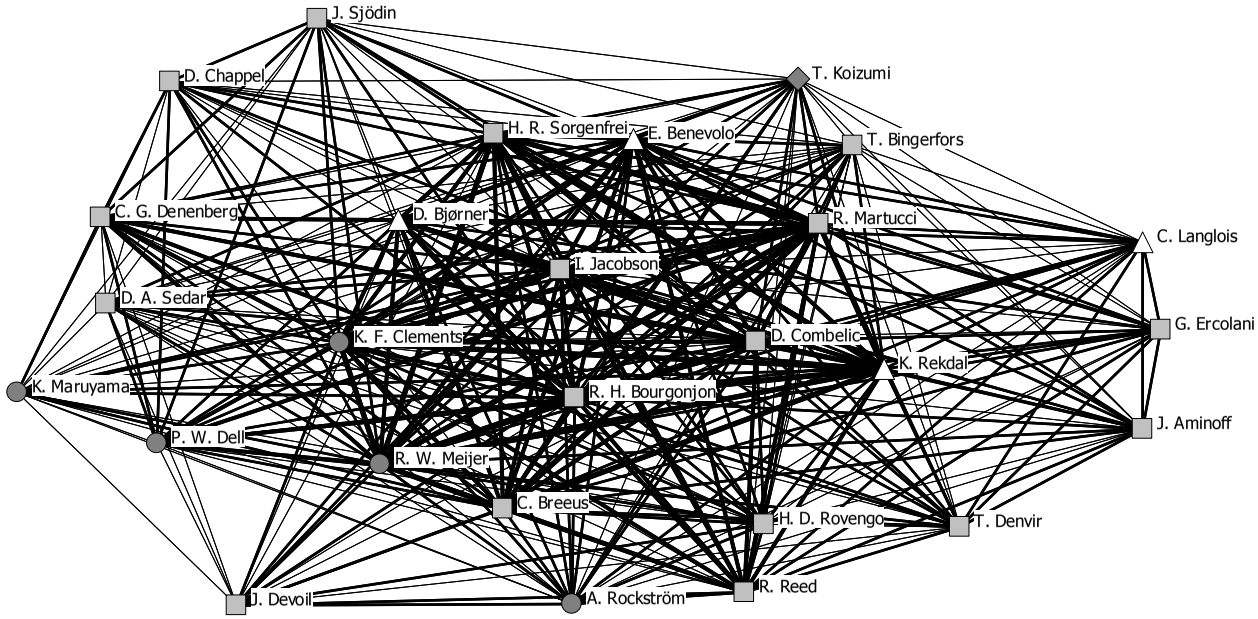


**Figure 4.2 Frequent participants in the Implementors' Forum.**[21]

[21] The individuals listed in the figure participated together with the individuals to which they are linked in at least three meetings. The complete network consists of 70 individuals. The width of the links illustrates the number of meetings in common to the participants. Dark grey circles denote participants from telecommunication administrations, while white triangles are representatives of research institutions or other academic institutions. Light grey squares denote manufacturing firms. A diamond indicates an unclear or unidentified affiliation status. Figure created using Netdraw. See Borgatti, NetDraw: Graph Visualization Software.

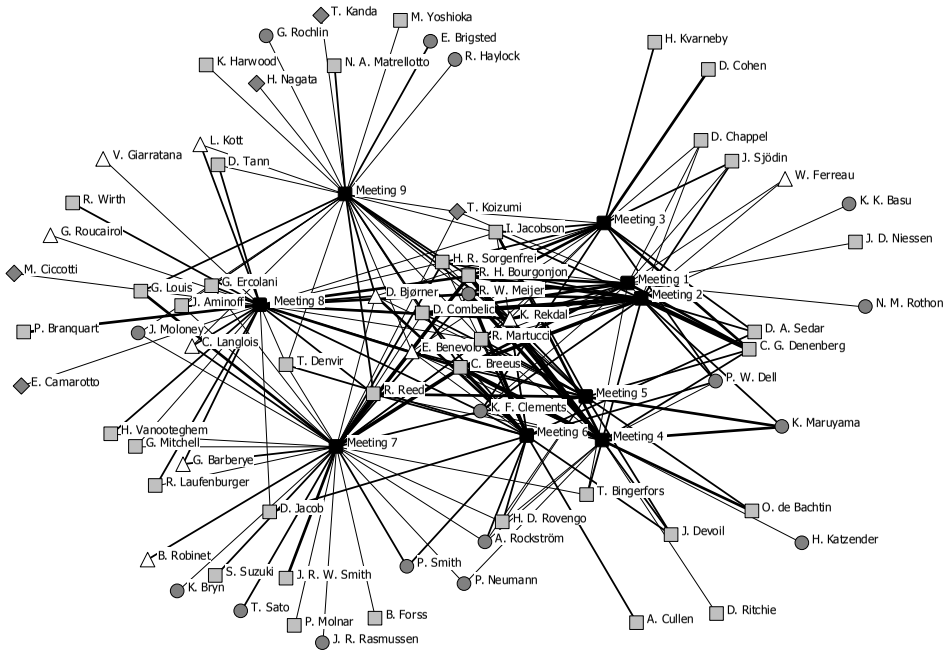


**Figure 4.3 Participants and meetings in the Implementors' Forum.**[22]

[22] The width of the links illustrates the number of written contributions made by an individual who also participated at the meeting. Dark grey circles are administration participants; white triangles are research institution participants and light grey squares are telecommunication administration delegates. The position in the diagram illustrates the measured centrality of the participant and the meeting, by the spring embedding technique. Figure created using Netdraw. See Ibid.

| | | *Co-appearance* | | |
|---|---|---|---|---|
| **Name** | **Organization** | **Degree** | **Degree (Alpha)** | **Degree (norm.)** |
| R. W. Meijer | PTT Netherlands | 192 | 320.28 | 1.000 |
| K. Rekdal | RUNIT | 192 | 320.28 | 1.000 |
| R. Martucci | S. I. T Siemens | 192 | 320.28 | 1.000 |
| C. Breeus | Philips (MBLE) | 176 | 285.25 | 0.889 |
| H. R. Sorgenfrei | GEC | 174 | 280.41 | 0.889 |
| R. H. Bourgonjon | Philips | 173 | 275.94 | 0.889 |
| K. F. Clements | UKPO | 165 | 271.37 | 0.889 |
| E. Benevolo | CSELT | 159 | 248.68 | 0.778 |
| D. Combelic | ITT | 158 | 246.34 | 0.778 |
| I. Jacobson | LME | 142 | 220.30 | 0.778 |
| R. Reed | GEC | 143 | 218.95 | 0.667 |
| D. Bjørner | Tech.Uni. Denmark | 141 | 210.94 | 0.667 |
| T. Denvir | ITT | 108 | 149.98 | 0.444 |
| C. G. Denenberg | ITT | 84 | 136.10 | 0.556 |
| P. W. Dell | UKPO | 86 | 132.92 | 0.556 |

**Table 4.3: Top 15 by joint appearance, sorted by the parameterised alpha-measure.**

Comparing the normalised degree centrality scores with the ones of the previous study period, it is evident that in real numbers, the well connected were a much larger group in the Team. 16 members of the Team scored above 0,5 on normalised degree centrality, while six comparable members were active in the period of the Team. Proportionally, 16 out of a total number of 70 delegates, was, however, a far smaller share.

| Name | Organisation | *Influence* Degree | Alpha |
|---|---|---|---|
| C. Breeus | Philips (MBLE) | 891 | 3249.22 |
| R. H. Bourgonjon | Philips | 838 | 2941.79 |
| R. W. Meijer | PTT Netherlands | 697 | 2215.26 |
| K. Rekdal | RUNIT | 521 | 1431.63 |
| C. G. Denenberg | ITT | 368 | 1247.95 |
| R. Reed | GEC | 423 | 1113.90 |
| D. Combelic | ITT | 361 | 850.75 |
| H. R. Sorgenfrei | GEC | 335 | 749.08 |
| K. F. Clements | UKPO | 324 | 746.71 |
| I. Jacobson | LME | 296 | 663.00 |
| D. Bjøner | Tech. Univ. Denmark | 281 | 593.46 |
| G. Louis | Philips (MBLE) | 251 | 592.79 |
| K. Maruyama | NTT | 209 | 581.48 |
| R. Martucci | S. I. T Siemens | 247 | 467.33 |
| J. R. W. Smith | GEC | 170 | 380.13 |

**Table 4.4: Top 15 by willingness to influence, sorted by parameterised alpha-score.**

When analysing the scores that measure document contributions, the team of Camille Breeus and Remi Bourgonjon, both from Philips (MBLE in the case of Breeus) emerges as the most active in the Forum. All in all, the measures indicate a very strong Dutch group, as the administration representative Meijer emerges in the top three in both tables, while Breeus and Bourgonjon score particularly highly when considering document contributions.

In the previous period, the working documents were typically from Philips, Runit and the NTT (with some notable exceptions). Working documents issued in the Forum included contributions from 20 different organisations, ranging from the Italian research institute CSELT to the Finnish telecommunication administration (the full listings are available in the appendix). All in all, the number of representatives of manufacturing firms was still the largest, just as in the Team of Specialists, but with a larger number of researchers with ties to administrations attending and influencing the direction of the work.[23]

Co-authored documents indicated an alliance between the participants in the forum. These alliances were important in the Forum, more so than in the Team of Specialists. 17 of 164 working documents were co-authored by delegates from different organisations. This type of contribution strategy was used by the veteran players, such as the participants from the Nordic telecommunication administrations, Philips, the Dutch administration and Runit. The main actors in the Forum in terms of activity were the representatives of Philips, who contributed to 34 working documents, either

[23] For the complete listing, see Appendix 2.

as sole authors or in collaboration with other contributors. Remi Bourgonjon also issued 16 documents as the convenor of the group.

The contributed documents can be grouped into three categories. One type dealt with inconsistencies of the language proposal and correcting errors, large and small. Another type was concerned with refinements of the language, often dealing with issues that had been postponed within the Team of Specialists or which they had left in a fairly preliminary state. Together, these two groups of documents were concerned with *programming language design*. A third group of documents were reports of various implementation projects, often reporting on various compilation design techniques.

Most documents concerning new features, refinements and correction of errors were submitted by organisations that had been active in the Team of Specialists. The latter group of documents, reports of progress in ongoing implementation projects, were issued by various participants. All in all, most contributions were issued by organisations that had been involved in the project from its start.

The idea behind a forum of *implementers* was to let practical experience feed back in to the language proposal. Practical experience with the programming language was mainly gained through the creation of compilers. The Implementors' Forum was extended by of a set of local initiatives and networks concerned with specific compiler construction projects. The Forum functioned as a central hub where information about the progress of local projects was reported, commented on and dealt with. There was much local variation in terms of host machines, target machines, level of commitment, competence, and investments. This variation was important for the development of the programming language itself, since a lot of this feedback pointed out ambiguities and errors in the language. Much activity in the Forum was dedicated to the issues that were raised when creating the compilers, which is the subject of the next few pages.

## Compiling Chill

The development of compilers, programs capable of translating code written in Chill into machine code understood by various telecommunication switches and computers, was a large part of what went on in the Implementors' Forum. 11 different implementation projects were embarked upon during the four-year period of the Forum, although not every one was carried out or could be described as successful.[24] Some of the projects were very limited in scope, producing nothing but a compiler for a small subset of the programming language to a specific hardware platform. Others were large and ambitious as they tried to design flexible compilers that could

[24] An overview is found in *Chill Bulletin* 1, no. 1 (1981), 39. This gives a status as of August 1981, a year after the official endorsement of Chill by the CCITT.

accommodate different hardware targets and host machines. Six projects were started at the outset of the Implementors' Forum. Siemens, ITT, Philips, the Nordic telecommunication administration, the Dutch administration and the NTT all started on constructing new compilers immediately. Others embarked on similar undertakings as the Forum period progressed. An overview of the compiler projects, large and small, is given in the table below. The compilers listed were either finished or under development as of August 1981. Projects that never got off the planning stage have been left out. Host computer implies the computer that ran the compiler and the target computer implies the computer or processor type for which the compiled code was intended. In this period of trials, the target computer did not always imply a computer platform used in switching products, but could be a general computer architecture, like Intel's revolutionary 8086 system introduced in 1978, which was the target of four of the projects.[25]

[25] Intel's 8086 processor holds a special place in computer history, as it eventually resulted in 30 years of successful chip designs that have allowed Intel to dominate microcomputers, like the 386, 486 and Pentium processors. On the history of Intel's 8086, see Stanley Mazor, "Intel's 8086", *IEEE Annals of the History of Computing* 32, no. 1 (2010).

| *Participating organsations* | *Host computer* | *Target computer* |
|---|---|---|
| Runit, The Nordic and British telecommunication administrations | Nord-100 | APZ-210,<br>Intel 8086 |
| NTT | DC-10 | DC-10 |
| ITT | IBM 370 and IBM 3033 | Intel 8086 |
| Siemens | Siemens 7000, IBM 370 | S7000, IBM 370<br>SSP103, SSP303<br>Intel 8086 |
| CSELT, SIP, Italtel | PDP 11/VAX | PDP11/VAX, PDP11/MIC-20 |
| Technical University of Denmark and the Danish telecommunication administration | VAX-11 | VAX-11 |
| CNET (French PTT) | IRIS 80 | Unknown |
| Philips | DEC-20 | TCP 36, TCP 16/Z 8000 |
| GTE | IBM 3033 | Intel 8086 |
| Dutch PTT | DEC-10, DEC-20 | PDP-11 |
| British Telecom | ICL 2900 | GEC Mark II BL |

**Table 4.5: Implementation projects started in the period of the CCITT Implementors' Forum.[26]**

How extensive were these projects? Economic details about the extent of investments in the implementation projects in terms of man-years or real sums are available only through sporadic evidence and casual reports of such to the Implementors' Forum. We know, for example, that ITT invested about five staff-years in their initial compiler project.[27] Other estimations indicated that Siemens, which in conjunction with the development of a Chill compiler also developed high-level debugging tools, had devoted in the region of 13 to 20 man-years to their early implementation project. Philips, which had been the most active manufacturing firm in the project overall, was supposed

[26] Information primarily found in various reports circulated in the Implementors' Forum, all documents found in KRC. On the compiler constructed by the Dutch PTT, see R. W. Meijer and G. H. te Sligte, "Status report of CCITT HLL implementation at the Dr Neher Laboratory of the Netherlands PTT", in *Software Engineering for Telecommunciation Switching Systems* (Helsinki, Finland: Institution of Electrical Engineers, 1978). On the Nordic Chill compiler, see Rekdal, "The Nordic CHILL Project". On the NTT, see K. Maruyama, N. Sato, and K. Konishi, "NTT CHILL implementation aspects and its application experience", in *Software Engineering for Telecommunciation Switching Systems* (University of Warwick, Coventry: Institution of Electrical Engineers, 1981). On the ITT compiler, see C. G. Denenberg, "CHILL Implementation techniques", in *Software Engineering for Telecommunciation Switching Systems* (Helsinki, Finland: Institution of Electrical Engineers, 1978).

[27] Denenberg, "CHILL Implementation techniques".

to have financed about 10-12 man-years for their initial implementation project.[28] The Nordic compiler project was perhaps the most wide-ranging, at least in terms of participating organisations. It involved the telecommunication administrations of Denmark, Sweden, Finland and Norway, and the Norwegian research establishment Runit. However, in terms of manpower, this project actually amounted to little more than 5-6 man-years, being fairly modest when compared with the industry-sponsored projects reviewed above. The Danish telecommunication administration also sponsored another project. The more sporadic projects, like compiler projects embarked upon in Italy and in France, were far more limited, even though details are hard to come by.

Summing up, the implementation projects embarked upon in the period of the Implementors' Forum were characterised by a small number of extensive and ambitious ventures, particularly led by firms that had already committed themselves to the cause through their active participation in the design phase, like the ITT, Siemens and Philips.

## Nordic cooperation and competition

Many of the compiler projects were organised within one firm or body, but one project tried to mimic the cooperative spirit of the Implementors' Forum at a regional level. The Nordic Compiler project was formally organised within the framework of the Nordic telecommunication conference, which initially was a biannually meeting of the administrations of the five Nordic countries. From 1969, the conference had broadened its mandate and organisation to a set of steering committees cooperating on technical matters.[29] Delegates of the telecommunication administrations in the Nordic countries would meet in a special working group from 1977. They met nine times until 1979 to follow up on questions regarding the Nordic compiler project and general Chill matters. The group worked well into the 1980s with further meetings and ultimately organising conferences about programming telecommunication equipment.[30] A considerable amount of information circulated among the groups' seven regular members and their parent

[28] All these estimates are based on information found in Knut Bryn, "Report of meeting no. 7 in Trondheim", 29/6 1979, NTT 77-2 Report no. 7, box "NTT 77-2 1977-1979", KRC.

[29] On the Nordic telecommunication cooperation, see Ari T. Manninen, "Elaboration on NMT and GSM Standards" (Univeristy of Jyväskylä, 2003), 39-41.

[30] See the proceedings of the NT-P Symposium on Languages and Methods for Telecommunications Applications, Turku, Finland, 6 – 8 March 1984, box "L 0135, Samarbeid," series "Da, 1961 – 1996", NTR.

organisations.[31] From 1978, the British Post Office joined the Nordic ranks and participated in the compiler project, strengthening the impression of a project oriented towards the administrations and the software knowledge they needed.

The administrations were supposed to create code tests and examples and carry out educational programs. This was something that was done with varying degrees of commitment, as the administrations in Sweden and Denmark were the most active ones, while the Norwegian administration participated in the code trials only in a very limited sense. The Norwegian research establishment Runit was responsible for the creation of the compiler and assisting the administrations with test runs of their code examples. However, the Nordic compiler project was not exclusively geared towards the administrations. At first, Runit was to cooperate with L. M. Ericsson and produce a compiler for their computer APZ-210, which was the computer used in the hugely successful AXE switch.[32]

The APZ-210 had been developed by Ericsson and the Swedish telecommunication administration. It was a "natural" target for the Nordic cooperation on Chill compilers, because of its Nordic origin, its ties to both Ericsson and the Swedish administration and because it was considered technically very advanced. However, the target for the Nordic compiler project changed as it soon became evident to the developers at Runit that the APZ-210 was so peculiar and esoteric that very close cooperation with L.M. Ericsson was needed. As technical information about the processor was hard to come by and no test machine was regularly made available to the Nordic group, the project never took off.

Initially, Runit's compiler was hosted on computers produced by the Norwegian computer manufacturer Norsk Data and the compiled code was transferred by modem to be tested out on an APZ-210 in Sweden. The participating administrations could submit their code examples for compilation on Runit's machine through a similar setup. However, the transfer of the test code was cumbersome and unsatisfactory.[33] Then Runit, in accordance with their sponsoring administrations, changed their target to a

[31] On 9 June 1980, an overview of all documents, publications and correspondence that had been circulated among the members of the Nordic working group, since its inception in 1975, was published. Here, 414 documents were listed. See Chill dokumentoversikt, 9 June 1980, box "NTT 77-2 1979-1980", KRC.

[32] The AXE system was developed by a joint research and development company called Ellemtel, owned by both L. M. Ericsson and Televerket, Sweden's state-owned PTT, from 1970. On the development of AXE, see Fridlund, "Switching Relations and Trajectories: The Development Procurement of the Swedish AXE Switching Technology"; Bengt-Arne Vedin, *Teknisk revolt: Det svenska AXE-systemets brokiga framgångshistoria* (Stockholm: Atlantis, 1992).

[33] Kristen Rekdal, interview with author, 28 November 2007, Oslo, Norway.

general Intel 8086 architecture, which had been released on the market in 1978. This made the compiler project of interest to other manufacturers, like the Swiss company Hasler and the Norwegian ITT subsidiary STK, which both planned on using the Intel architecture in future switching equipment. STK and Hasler started to invest in the project by issuing development contracts to the Runit group, to which I will return in chapter seven. Here it is sufficient to note that the Nordic compiler project appeared interesting and viable to the industry already at its inception.

The two most active participants in the initial Nordic compiler project, beside the contractor Runit, were the Swedish and the Danish administrations. The latter sat up a local contact group that coordinated the Danish participation in the Nordic project and carried out experiments at a local exchange in Kolding.[34] The feedback was not all positive or supportive. In Denmark, there was a distinct reaction from some of the electrical engineers involved in the project that the language was too far removed from conventional telecommunication practices. Two members of the contact group, the engineers Ove Færgmand and Jørn Johansen, both of the telecommunication research establishment (TFL) run by the Danish telecommunication authority, and the three Danish telecommunication operators, all expressed views that the language was not satisfactory. Too little effort, according to Færgmand and Johansen, had been spent on practical problems like "reliability and error messages".[35] Such impressions were not uncommon. Remi Bourgonjon has, in retrospect, referred to a similar type of response in Philips:

> I remember that, when the first CHILL language documents became available, a manager at my company, experienced in telephony applications, was very disappointed with the result. He had expected that a telephony language, as CHILL was announced to be, would have statements such as "switch path from A to B" and "give ringing tone".[36]

These arguments give a good impression of the encounters between the language designers and the potential users (typically electrical engineers), and that they were not always straightforward. The expectations of what a programming language should be were widely different. Typically, issues like reliability and the possibility of portable code were high on the agenda of many administration-employed electrical engineers.

---

[34] Jens R. Rasmussen,"Mødereferat, 4 april 1978", box "NTT 77-2 (1977-1978)", KRC.
[35] Ibid.
[36] Remi Bourgonjon, "Programming languages, Environments and CHILL," *Chill Bulletin* 3, no. 1, (1983), 3 – 8.

The Danish contact group not only mediated interests between the Runit project and the Danish authorities. It also followed a separate Danish compiler project started at the Technical University of Denmark. This project was led by Dines Bjørner and was carried out by the Technical University of Denmark and the Research Laboratory of Telecommunication. From the outset, this project relied heavily on a formal and mathematically oriented approach to compiler design. This was a natural to Bjørner, who was a professor at the Technical University of Denmark, but had working experience at the IBM Vienna Lab from 1973 to 1975. This laboratory spearheaded the development of formal development methods and definitions during the late 1960s and throughout the 1970s. In particular, the years when Bjørner was actively involved with the work in Vienna have been considered of great importance when it comes to the history of formal programming language descriptions.[37] For example, the very first formal definition of programming language semantics was created for the programming language PL/1 (which was created by the IBM) at the laboratory in 1974.[38] As such, it should be of no great surprise that Bjørner's project was strictly focused on formalism: he applied what was known as the Vienna Development Method (VDM) to the Chill compiler project, a method he had partly developed when based in Vienna. To Bjørner, the ideal was that formal definitions of a programming language should presuppose the compiler design:

> We believe, seemingly contrary to all textbooks on compiler design that the very initial stages of any compiler development must concentrate first on a precise description of the source language and the target language, to be followed by a precise description of the compiling algorithm.[39]

---

[37] An overview is found in Kurt Walk, "Roots of Computing in Austria: Contributions of the IBM Vienna Laboratory and Changes of Paradigms and Priorities in Information Technology", in *Human choice and computers: Issues of Choice and QUality of Life in the Information Society*, ed. Klaus Brunnstein and Jacques Berleur (Dordrecht: Kluwer Academic Publishers, 2002).

[38] D. Bjørner and C. B. Jones, *The Vienna development method : the Meta-language*, Lecture notes in computer science 61 (Berlin ; New York: Springer-Verlag, 1978).

[39] Dines Bjørner, "Programming Languages: Formal Development of Interprenters & Compilers", in *International Computing Symposium*, ed. E. Morlet and D. Ribbens (Liege, Belgium: North-Holland, 1977). Bjørner's contribution was one of a few invited papers to the conference, and was published alongside a paper by Edgser Dijkstra entitled "Programming: From Craft to Scientific Discipline", a title that gives an idea of the issue at stake. See Edsger W. Dijkstra, "Programming: From Craft to Scientific Discipline.", in *International Computing Symposium*, ed. E. Morlet and D. Ribbens (Liege, Belgium: North-Holland Publishing Compnay, 1977).

These precise descriptions should be mathematical in character, and should make it possible to prove correctness, again according to Bjørner:

> First, and independently of each other, one must have a precise, unquestionable, terse and formal definition of the source (CHILL) and target (IBM series/1). Formality is required so that one is able to prove properties of e.g. the CHILL compiler (like correctness!), CHILL programs (like their correctness!), and the runtime system on the IBM series/1 enabling single CHILL programs, consisting of multiple co-ordinated processes, to be scheduled, to share logical resources, to be allocated physical resources, and to communicate and be synchronized.[40]

Bjørner's approach contrasted substantially with that of the other projects, by not being geared towards the practical use of the finished compiler but towards proving the applicability of the formal methodology to compiler design more in general. The electrical engineers of the Danish telecommunication administration would not encounter practical trials and a focus on reliability and errors in the Danish compiler project. Instead, they came up against semantic formalism. None of the other compiler projects approached the issue of compiler design with the scrutiny of the team led by Bjørner. Compiler designs more concerned with the performance of the compiler and the effectiveness of the compiled code, rather than its correctness, were typical. One such example was the efforts led by the ITT, which is the subject of the next section.

## Compilation in the ITT

One substantial compiler project developed by the ITT is worth noting. The ITT project had a large geographical scope, because of the freestanding company's multinational character: in the 1970s ITT had operations in 10 countries (not including the United States) that were concerned with the development and/or supply of telecommunication switching systems.[41] The ITT's advanced technology centre in Connecticut had the mandate to create a compiler that could take the considerable diverse needs of ITT's subsidiaries into account. The research centre had been set up in the mid-1970s, with the main task of developing the System 12 switch. The compiler development was part of the same project, even though the compiler was intended to service multiple ITT installations, and as such had to cater for a variety of target machines as well as being portable between host machines. The particular needs of the ITT were communicated in the following manner to the Implementors' Forum:

---

[40] Dines Bjørner, "The ID/L/CHILL Project – An overview", April 7 1978, SJ1, KRC.

[41] C. G. Denenberg, "Chill Implementation techniques." box "CCITT IF Arbeidsdokumenter 6, Serie J", KRC.

> Particular consideration was given to questions of compiler portability to different host machines, code generation for different target machines, the mixed usage of CHILL and ESPL-1 (The ITT HLL for SPC) in existing SPC system developments, and compiler maintenance at various locations. These questions of technology transfer influenced both the project objectives and the final compiler design.[42]

While a formal approach like the one proposed by Bjørner could be understood as a method towards portability, which also was needed in the ITT, this project was much more conventional in its approach than that of Bjørner's team. The ITT focused on designing what was essentially a modular compiler with what could be described as an interchangeable back-end, which ITT hoped would cater for the different needs of the various subsidiaries. The design decisions implied a compiler designed in a way that should enhance its portability, as the interchangeable back-end would cater for different hardware platforms, while retaining the Chill-specific front-end.[43] However, the in-house design of a compiler by the ITT's advanced technology centre did not produce an effective compiler. In around 1980, it was decided that a more effective compiler, in terms of the object code it produced, had to be developed. This time, the ITT hired an outside firm, the Massachusetts Computer Associates, to develop the compiler that would eventually produce the code for the System 12 switch.[44] The lofty goals of portability had previously got in the way of an effective code. According to Tom Love, formerly of the ITT and directly involved in the decision to contract for a new compiler outside the ITT walls, this piece of the system was crucial, as he noted: "Had a fast enough and correct enough compiler not been provided, the 1240 [the System 12] could not have succeeded."[45]

Summing up, the above analysis of compiler projects reveals the variety in organisational principles, commitment and technical approach to the issue. In terms of organisational form, the Nordic compiler project was special, as it consisted of a network between the research organisation Runit, the Nordic telecommunication administrations and some interaction with the manufacturer L.M. Ericsson. The more typical projects were the in-house development at Siemens, Philips and the ITT. In terms of technical approach, I have highlighted how the Danish project led by Dines Bjørner

---

[42] Ibid.

[43] On the development of compiler structure, see Aho and Ullman, *Principles of compiler design*; Wirth, *Compiler construction*.

[44] Here, I rely on email conversations with Tom Love (formerly of ITT), February 2011 and his short account of this in Tom Love, *Object lessons : lessons learned in object-oriented development projects*, Advances in object technology 1 (New York: SIGS Books, 1993), 81.

[45] Tom Love, e-mail to author, 15 February 2011.

mirrored a formal development virtue that characterised parts of European computer science, while the others were more geared towards performance aspects, on both host and target machines.

Problems that emerged locally when trying to design the compilers fed back into the language design processes in the CCITT group. This was particularly so in two areas: the implementation of language concepts of so-called concurrent programming and the creation of a formal definition of the programming language. The former was directly linked to programming language design, while the latter only implicitly so. Concurrency had popped up as a challenge in many of the implementation projects: ITT, for example, would spend a considerable amount of time working on their implementation of language constructs for process handling in their implementation project.[46] Formal descriptions were, in particular, related to the work done by Dines Bjørner and his team. It was first and foremost understood as a part of their take on compilation – and only implicitly associated with the design of the programming language. The formal description was believed to make it easier to construct compilers, and could also help in getting rid of inconsistencies in the language itself by explicating relationships that were difficult to come by through other descriptions.

In the following pages, I will account for both of these issues, which dominated much of the activity in the Forum during this period. Both cases highlight how decisions made in the Implementors' Forum were a continuation of the alliances between actors subscribing to what I have called a relatively formal and mathematically oriented development virtue, regardless of whether they had a background in computer science or the emerging field of software engineering.

## Concurrent processes and decisions

Concurrency drew a lot of interest from computer scientists from the mid-1960s and onwards, and is again receiving renewed attention nowadays.[47] Basically, concurrency means parallelism. At the machine level, operations are sequential if they occur one after another in time. Operations are concurrent if they overlap in time. At the software level, concurrency involves the notations for expressing potential parallelism so that operations may be executed in parallel at the machine level. Concurrent programming languages are programming languages that use language constructs for

[46] C. G. Denenberg, "Chill Implementation techniques," box "CCITT IF Arbeidsdokumenter 6, Serie J", KRC.

[47] As a result of multi-core processors gaining in popularity in regular PCs, concurrency has become a major issue in contemporary programming language discussions. For a casual overview, see the interviews in Federico Biancuzzi and Shane Warden, eds., *Masterminds of Programming* (Beijing: O'Reilly,2009).

execution of computational operations in parallel, rather than in sequence. These constructs may involve concepts such as multi-threading, support for distributed computing, message passing and shared resources such as memory.[48] Furthermore, they include concepts for passing information from one concurrent process to another. Such capabilities would be an obvious boon to a language intended for usage in a real-time system such as telecommunication switching, which almost per definition involves parallelism, or to put another way, the real world phenomena with which the telecommunication software was to engage was concurrent by nature. One may think of the thousands of calls switched at the same time in such a system to understand this.

The CCITT group started to consider concurrency seriously as they entered the implementation phase. Such concepts had also been discussed in the Team of Specialists, but were intentionally left for the Implementors' Forum to work on. The overall state of concurrent programming language design in the mid-1970s was an important reason for this.

According to pioneer Per Brinch Hansen, the first steps towards an understanding of concurrent programming were taken in the mid-1960s, but then developed fundamentally in the 1970s.[49] In 1971, Tony Hoare claimed that the search for language features that would allow for parallelism and concurrency was "one of the major challenges to the invention, imagination and intellect of computer scientists of the present day".[50] Concurrency had been achieved by employing a number of different techniques since the 1960s, but was not understood in any solid theoretical way before the 1970s. Several landmark articles had been published before the Team of Specialists was drummed together in 1974, but all in all, the efforts made in the Implementors' Forum to create language concepts that would allow for concurrency were made just shortly after the publication of important research on the subject matter, and in some aspects in parallel to fundamental theoretical research. The first concurrent programming language was in fact only developed in 1975, by the aforementioned Dane, Per Brinch Hansen. Furthermore, the first book on concurrent programming was only issued in 1977, the very same year that the Chill designers started discussing their interpretations of the issue.

It is worth noting that great scientific strides in this field were done by computer scientists who have been held in high esteem in computer science

---

[48] An overview of abstractions for concurrency that also discusses their implementation in Chill is J. M. Bishop, *Data abstraction in programming languages*, International computer science series (Wokingham, England ; Reading, Mass.: Addison-Wesley, 1986), 102-34.

[49] Hansen, "The invention of concurrent programming".

[50] Here quoted from Ibid., 16.

in general. When Per Brinch Hansen collected what he regarded as *the* classic papers in concurrent programming almost 30 years later their initial publication, he was surprised to see that every single paper turned out to have been written by Edsger Dijkstra, Tony Hoare or himself.[51] Dijkstra and Hoare have already been mentioned as influential actors in a move towards the mathematically oriented computer science. They also made particular important contributions to the development of concepts facilitating concurrent programming. According to Judy Bishop, their simultaneous interest in the topic was unrelated, but spurred by a similar set of agendas:

> The swing away from assembly language gained genuine momentum during the seventies was slow to affect the area of concurrent systems – operating systems, embedded control systems, and the like. What happened was that three people – Edsger Dijkstra, Tony Hoare and Per Brinch Hansen – independently developed key abstractions which were taken up by researchers worldwide, realized in experimental languages, reported on, adapted and refined. In this way, the problems of concurrency could be expressed in well understood notation, and solutions and principles gradually evolved.[52]

This was in full motion in the late 1970s, and informed the designers of Chill. In typical committee fashion, the issues of concurrency would take a considerable amount of time to resolve and involved a number of compromises affecting the finished language, not least because concurrent processing principles touched upon concepts quite familiar to those engaged with traditional telecommunication technologies.

The first proposal regarding concepts for concurrency in Chill was made by Charlotte Denenberg of the ITT, at the very first meeting of the Implementors' Forum in September 1977.[53] Following this "tentative agreements were reached upon this subject" at this meeting.[54] At the second meeting, in November 1977, Denenberg made further proposals.[55] A contribution from GEC and two from L. M. Ericsson were also put forward

---

[51] Ibid.
[52] Bishop, *Data abstraction in programming languages*, 103.
[53] C. D. Denenberg, "Proposed HLL Primitives for Process Handling", document number G4, RBC.
[54] CCITT HLL Implementors Forum, "Progress report on the first meeting, London, 12 – 16 September 1977", RBC.
[55] C. D Denenberg, "A higher level set of synchronization primitives", document number H7, RBC; C. D. Denenberg, "Additions to the process handling primitives", document number H8, RBC; C. D. Denenberg, "Comments on 'proposal for monitor-like feature'", H9, RBC.

at this meeting.[56] The two contributions from the Swedish firm sparked a long and winding discussion. L. M. Ericsson had not been particularly active in the development of Chill before the issue of concurrency came up, but they put a lot of weight behind their proposals on concurrency. This was related to the fact that L. M. Ericsson had previous experience in using one specific concurrency concept, often referred to as the signals concept, in their programming language Plex. Representatives of the firm had also been heavily engaged in moving this concept into the other language that was developed by the CCITT at that time, the specification and description language SDL.[57] At the time, pressure was also being put on the convenor of the Implementors' Forum, Remi Bourgonjon, to strive for harmonisation between SDL and Chill.[58] With L. M. Ericsson's entrance into the debate and the increased pressure towards linking SDL and Chill, diplomatic negotiations became complicated and were being conducted on multiple fronts. In general, the main contestants were a constellation of Philips, the Dutch administration and the Nordic delegates from Runit on the one side, ITT on another and the Swedish manufacturer L. M. Ericsson on yet another side. How they would eventually end up with a proposal that everyone was able to agree on was in itself a combination of concurrent processes and diplomatic adjustments.

An important participant in this debate was Ivar Jacobson, who started participating in the Implementors' Forum from their second meeting.[59] Jacobson had experience of hardware design and telecommunication software systems and applications development.[60] He had, however, no education in the field of computer science, nor any experience in programming language design or compiler programming. He described his own role as "a cat among ermines", the odd one out, when interviewed about his role in the CCITT group, and said this was because his background was different from that of many of the participants in the Implementors' Forum.[61] The representatives of L. M. Ericsson argued that the world of

---

[56] GEC, "Substring handling", document number H15, RBC; L. M. Ericsson, "Proposed HLL concepts for process handling and interprocess communication", document number H17, RBC; L. M. Ericsson, "Proposed addition to the HLL for process handling and interprocess communication", document number H18, RBC.

[57] Ivar Jacobson, telephone interview with author, 22 February 2011.

[58] Remi Bourgonjon, telephone interview with author, 21 March 2011.

[59] CCITT HLL Implementors Forum, "Progress report on the first meeting, London, 12 – 16 September 1977", RBC.

[60] Ivar Jacobson had, by 1978, been working on the AKE and AXE switching systems as what can be called a software architect, for about 10 years. On his background, see Vedin, *Teknisk revolt: Det svenska AXE-systemets brokiga framgångshistoria* 101-06.

[61] Ivar Jacobson, telephone interview with the author, 22 February 2011.

telecommunications in particular, and electrical engineering in general, had something to teach computer scientists, programming language experts and software engineers, when it came to concurrency. In a working document by the participants from L. M. Ericsson, it was argued that telecommunication switching systems would need special attention and that the general techniques developed by computer scientists were insufficient.[62] Coming from this, the L. M. Ericsson representatives presented the Forum with their signals concept for inclusion in the language proposal. L. M. Ericsson's signals concept, however, met with scepticism. It was argued that the favoured Ericsson solution was too high level and abstract, and would tax the limited hardware resources too much.[63] According to Jacobson, the other participants would repeatedly claim that what Jacobson (and L. M. Ericsson) wanted to introduce to the standard was "semantically equivalent" to what they already had.[64] Regardless of these specific objections, it seems likely that the members of the Forum had a hard time grasping what Jacobson and L. M. Ericsson actually tried to get into the programming language, in particular the exact semantic meaning of the concept.[65] The different background of the contestants was a large part of the reason for this. By January 1978, Charlotte Denenberg, Ivar Jacobson and Remi Bourgonjon held an informal meeting in the Netherlands, to get an agreement on these issues.[66] The outcome of this meeting was a reconciled proposal that would include the ideas from Philips, ITT and the signals concept so much argued for by Jacobson and L. M. Ericsson.[67]

However, this compromise was not enough. The disagreements flared up yet again at the third and fourth meetings of the Forum. After a whole day of debating how to implement concurrent concepts in the language at the meeting of the Forum in June 1978, the group's vice-convenor, Kristen Rekdal, was pessimistic. He reported the following to his sponsors, the Nordic telecommunication administrations: "One whole day was devoted to the subject of concurrent processing. Sorry to say, no agreement could be reached this time either. There is still some hope, however, because the

---

[62] L. M. Ericsson, "Process intercommunication", document I-9 in appendix C in "Progress report on the third meeting, the Hauge, 13-17 February 1978", box "Arbeidsdokumenter 5, Serie I", KRC.
[63] Kristen Rekdal, email to author, 14 July 2009.
[64] Ivar Jacobson, telephone interview with author, 22 February 2011.
[65] Ibid.
[66] "Minutes of the informal meeting (23-25 January 1978) about the SDL semantics and concurrent processing concepts for the HLL", RBC.
[67] Remi Bourgonjon, "Language aspects of process communication/synchronisation", I4, RBC.

disagreements were smaller this time."[68] In particular, Jacobson was not content with the results of the informal meeting, nor the subsequent discussions in the Forum. L. M. Ericsson and Ivar Jacobson eventually found an unlikely ally in the concurrency debate, as they teamed up with Dines Bjørner. At the fourth meeting of the Forum, in June 1978, the new allies presented a new proposal.[69] Bjørner used the concurrency debacle as a way to prove the usefulness of formal descriptions, and by creating a formal description he really translated Jacobson's signals concept into a terminology that could be accepted by the other group member. However, L. M. Ericsson and the Danish administration developed a common proposal that completely sidelined the prior proposals from Bourgonjon, Denenberg and Jacobson.[70] Here, the representatives from L. M. Ericsson and Bjørner argued the following:

> The underlying ideas of the proposals are not new. Thus they e.g. resemble very closely the basic concepts of C. A. R. Hoares' "Communicating Sequential Processes." The authors of the proposal are also pleased to note a very striking one-to-one correlation to basic concepts of SDL. Whereas the buffer/semaphore based constructs have been around for more than 10 years, the currently proposed constructs appear to form a much needed replacement of these older ideas.[71]

This quote reveals a set of relationships that somewhat breaks down the dichotomous set of development virtues sketched out in the introduction: On the one hand, it signifies a link to cutting edge research on concurrency by aligning themselves with researchers such as Tony Hoare and Brinch Hansen, but at the same time identifying this route as something tied to electrical engineering thinking, as it notes the relationship to the description language SDL that was under development in the CCITT at the time. Furthermore, the authors noted that the existing proposals on the table in the Chill project were rather old fashioned, as the last line of their statement argued.[72]

How did the revised proposal go down with the other participants? The replacement strategy proposed by Bjørner and Jacobson did not succeed in its entirety. Still, L. M. Ericsson's favoured concept, the signals concept,

---

[68] Kristen Rekdal "The third meeting of the CCITT Implementors Forum," 20 February 1978, box "Arbeidsdokument 5, Serie I", KRC.
[69] Danish P&T, L. M. Ericsson, "Process communication in CHILL; a set of proposals", J3, box ""Arbeidsdokumenter 6, Serie J", KRC.
[70] On the second page of the proposal, this is made explicit: "The set of proposals should be seen as a complete, substitute replacement [...]." See Ibid, 2.
[71] Ibid, 2.
[72] Ibid, 2.

had crept its way into the language recommendation, already before the joint proposal by Bjørner and Jacobson. The final recommendation would still look similar to the one sketched out before Bjørner's intervention to formalise the signals concept. However, according to Ivar Jacobson, the Bjørner-led definition was crucial to get the signals concept properly understood by the other members.[73] To reach an agreement, the committee would still have to arrive on a compromise by the method of an informal meeting: this time, the agreement was reached at an informal meeting between the actors in the late summer of 1978. Remi Bourgonjon, Ivar Jacobson, Kristen Rekdal, Oleg de Bachtin and Anders Rockström were the participants. Here, they finally came to an agreement on harmonising the concurrency concepts agreed on at the previous Forum meetings.[74] The results of the meeting, and its reception in the Forum, made Kristen Rekdal optimistic. He reported the following after the subsequent meeting in the Forum:

> Again [concurrency] consumed a large part of the meeting. This time there was luckily, agreement on most of the semantics, and even a provisional syntax. The agreement largely follows the document submitted by RUNIT, LME and Philips. It is hoped that the topic can now be settled at the next meeting.[75]

The settlement was, in other words, a hybrid. It did not opt for the complete revised proposal that Bjørner and Jacobson brought to the table, but Jacobson's favoured signals concept got into the final recommendation, perhaps in a more properly understood manner than without the alliance between Bjørner's formalism and Jacobson's ideas.

Would such a hybrid degrade the result? In retrospect, this aspect of Chill has been regarded as one of its real strengths. Kristen Rekdal, who initially had been sceptical about the concepts, has stated that he is happy it was added, and that it became one of the most powerful constructs in the language. Furthermore, with the great strides made in hardware design and capacity, the feared lack of efficiency was largely overcome.

All in all, the concurrency debate highlights the way compromises were not always such a bad thing. Kees Smedema, which participated in the CCITT work on Chill in the early 1980s on behalf of Philips, stressed this point:

---

[73] Ivar Jacobson, telephone interview with author, 22 February 2011.

[74] Noted in O. de Bachtin, R. H. Bourgonjon, I. Jacobsson, K. Rekdal, A. Rockström, "Review of Concurrent Processing", K5, box "Arbeidsdokumenter 7, Serie K", KRC.

[75] Kristen Rekdal, "The fifth meeting of the CCITT Implementors Forum, Tokyo 18-22 September 1978", box "Arbeidsdokumenter 7, Serie K", KRC.

> [Chill] was designed by a committee, which has the well known disadvantage of having to compromise between the various preferred concepts of committee members, resulting in non-orthogonal language design. However, it has also an advantage: the experience of many people is shared. In the case of designing the facilities for concurrency in Chill it was definitively a great advantage that the people involved were experienced in software for switching systems […].[76]

## Formal definition

In parallel to the concurrency debate, some members of the Implementors' Forum started work on a more formal definition of the semantics of the language, meaning a model describing the possible computations by the language. Such a definition would add to the formal syntax and semiformal semantics already produced, and would create a description of each phrase in the language in some sort of other language, usually through mathematical formalism rather than another computer language.

Again, the main inspiration as well as the initiative came from actors rooted in computer science. An important catalyst was Dines Bjørner. To Bjørner, Chill was already too complex and riddled with compromises.[77] One way out of the quagmire was to impose a so-called formal denotation of the semantics of the language. To Bjørner, such a definition of the language could ease its way towards the finished article, into the real world of language *implementation and compilers*. This was particularly so since such a formal definition of the language semantics could help *prove* that a particular compiler implementation was correct. It would also reconcile Chill with the mathematical and formal virtues that dominated the constituencies of computer science.

Bjørner had started work on a Chill compiler prior to his involvement in the Implementors' Forum and had experienced that the design of a Chill compiler could be helped by the use of what was known as "denotational semantics," a field Bjørner had considerable knowledge of from his involvement with the IBM research laboratory in Vienna in the early 1970s. The outcome of Bjørner's effort would go well beyond the limited problems of compiler construction, as it spun off an effort to formalise the language semantics within the Implementors' Forum. The aim of denotational semantics was to formalise the programming language by constructing mathematical objects (called denotations) that would describe the meanings

---

[76] C. H. Smedema, "CHILL: Facilities for Concurrency", in *COMPSAC* (Chicago: IEEE Computer Society Press, 1983).

[77] Dines Bjørner and Peter L. Haff, "A formal 'denotational' semantics definition of CHILL", Technical report ID888, September 14, 1979, ix. Available in box "NTT 72-2/NT-P 1979 – 1980", KRC.

of expressions from the languages – a large step towards full formalism from the more common route of formally defining the syntax of a language.

There were high hopes. In 1978, it was noted that the denotational semantics would be "a very valuable contribution to the development of CHILL".[78] Some devotion was put towards formal definitions in 1979, hot on the heels of the agreement reached on concurrency by the detour of formal definition description of L. M. Ericsson's signals concept. The efforts were pushed forward by two parties, one group led by Dines Bjørner and another that involved researchers at the Belgian Philips laboratory of research, the MBLE, primarily Paul Branquart, George Louis and Paul Wodon. The latter group had already been involved with the work in the Team of Specialists, through their affiliation with Philips, and held substantial experience of work on the programming language Algol 68. After a short period where both definitions competed for the attention of the Forum members, the MBLE group decided to continue their work as an aid for the compiler group of Philips, and not to pursue the larger goals of Bjørner's group.[79]

Bjørner hoped that the formal definition would "gain confidence in the language design, to check its completeness and consistency".[80] Bjørner expressed a sincere belief "that with the advent of this formal document a basis has been established for an orderly control of future developments of the language".[81] As such, it was a tool to minimise the ambiguity of the complex language, but also a tool that could point out deficiencies in the casually described language. This was not only something that Bjørner believed, but a view shared by many of the members of the Implementors' Forum. As many discussions in the group were oriented towards ambiguities in the existing and at best semi-formal description of the semantics, it was believed that some of the ambiguities could have been avoided to some degree with a formal description.[82]

Bjørner's approach was rooted in a belief that a disciplined software development methodology, based on proof and formalisation, was a

---

[78] Kristen Rekdal "The third meeting of the CCITT Implementors Forum", 20 February 1978, box "Arbeidsdokument 5, Serie I", KRC.

[79] The formal description created by the Belgian group was published by Springer Verlag in 1982, as part of their Lecture Notes in Computer Science series. See Paul Branquart, Georges Louis, and Pierre Wodon, *An Analytical Description Of CHILL, the CCITT High Level Language*, ed. G. Goos and J. Hartmanis, Lecture Notes in Computer Science (Heidelberg: Springer-Verlag, 1982).

[80] Dines Bjørner and Peter L. Haff, "A formal 'denotational' semantics definition of CHILL", Technical report ID888, September 14, 1979, i.

[81] Ibid.

[82] Confirmed by Kristen Rekdal in e-mail to author, 14 July 2009.

necessity.[83] Software development should be drenched in the virtues of computer science. The formal definition was not ready to be published as a supplement to the official recommendation. Instead, it was published in 1981, as a result of repeated delays. As such, the arrival of the formal definition was too late to influence or help the budding compiler designers, who would be the main users of such a definition.[84] In this respect, the high point of the influence of the mathematical development virtue became nothing but a delayed supplement to the official CCITT recommendation. This was due to the extreme complexity involved in creating and understanding such a definition. It was full of rigour and clarity, but it was not something that was well understood in the higher ranks of the CCITT hierarchy, or in telecommunication organisations. Still, with the formal description, Chill became the first industrial and widespread programming language to have, as part of its recommended standard, a rigorous definition. The result, however, was a disappointment. According to Bourgonjon, all forms of formal descriptions have a drawback: "There are several tools known to formally describe the context-free syntax, the static semantics, and the dynamic semantics [....]. Unfortunately, all those tools are hampered by the fact that they are very difficult to understand for non-experts."[85] When discussed in hindsight, Remi Bourgonjon argued that the formal description also did not help in vindicating errors in the language proposal: "The formal definition did not help anyone. I, for one, found the same errors in the formal definitions as I found it in the semi-formal description."[86] He followed by adding that he often found errors, other than those already noted by the Danish team, by applying the semi-formal approach.[87] It also proved to be very difficult and time consuming to verify the equivalence between Z.200 and the formal definition. This could only be done by experts on both Chill and VDM carefully and meticulously reading both documents side by side.

---

[83] Kristen Rekdal wrote that "VDM is more formal than BLW" in "Informal meeting on "CHILL Formal definition, Hilversum 22 – 23 March 1979," box "Implementors Forum, serie M", KRC.

[84] The definition was issued as a manual to the Recommendation Z.200, but was only made available on request. Its prelude was published in the Chill Bulletin. See Peter Haff, "Chill Formal Definition", *Chill Bulletin* 2, no. 1 (1982), 11 – 22. It was also published as Peter Haff, "A Formal Denition of CHILL - A Supplement to the CCITT Recommendation Z.200", in *Technical Report* (Lyngby, Denmark: Dansk Datamatik Center, 1980).

[85] Remi H. Borugonjon, "The CCITT High Level Programming Language", in *Software Engineering for Telecommunciation Switching Systems* (Helsinki, Finland: Institution of Electrical Engioneers, 1978), 39.

[86] Remi Bourgonjon, interview with author, 16 January 2009, Heeze, the Netherlands.

[87] Remi Bourgonjon, telephone interview with author, 17 March 2011.

Even the maintenance of the formal definition proved impossible. Every time there was a change in the recommendation document, the formal description had to be updated and re-verified.

Even to the supporters of formal definition, the Chill report was lacking in some respects. According to the "official" VDM annotated bibliography, the report "features an attractive style of presentation of operational intuition but fails to have been given a satisfactory semantics".[88] Still, it was a starting point for more work on how to provide a formal description of concurrency, and thus spurred further refinements in the formal approach that the Danish computer science community were so fond of.

To Bourgonjon, formal semantics was one step too far. Progress towards a more unambiguous language was made through logical thinking and extensive collaboration rather than through the use of a formal denotational semantics and a formal development method. This highlights how the project was a true intersection *between* formally oriented computer science and more practical concerns, and how some ideas about strictness and formal languages did not pave out. In another way, one would believe that such a definition would enhance the image of Chill in the formal computer science camp. It did not. Chill continually met with scepticism or blatant ignorance from actors in the computer science camp.

It is worth noting that the cases of concurrency and formal definitions reveal two general mechanisms regarding the relationship between computer science and engineering in the development of Chill. Firstly, both implementations hinged on an active involvement of the computer science community: The implementation of concurrency was highly dependent on the applications of theoretical arguments advanced by influential scientists like Tony Hoare and Edgser Dijkstra. The application of formal descriptions was obviously another offspring of European computer science, highlighted by Dines Bjørner's prior involvement with the IBM research laboratory in Vienna. However, these applications of theoretical computer science were just as dependent on fitting in and realigning with the concerns of telecommunication professionals and electrical engineers. The case of L. M. Ericsson's relative high-level concepts for concurrency is most revealing: Here, the formal definitions of Dines Bjørner helped the idea translate into a vocabulary understandable and tolerable to the rest of the Forum's members.

I have interpreted these oscillations as processes of alignment of development virtues and ideals. However, it is also important to note that the process of creating a standardised programming language was a process of codification of practical experience and the assimilation of external pressure.

---

[88] Peter Gorm Larsen, "The VDM Bibliography", (Odense: The Institute of Applied Computer Science, 1996).

In the next sections, I discuss the externalities of Chill, in the form of stakeholders such as the telecommunication administrations and its programming language competitors, in particular the emergence of a standardised programming language for embedded real-time systems made for the American Department of Defense, Ada, and C, created at AT&T.

## Travelling in Chill

In September and October 1979, Kristen Rekdal travelled extensively around the world with what was to be the final reference manual of Chill in his suitcase.[89] First, he attended what was the last meeting in the Implementors' Forum in Melbourne, Australia. After that, he participated in the official CCITT meeting of the sub-working party XI/3, which officially nominated the work of the Implementors' Forum to the CCITT Plenary the coming year. Along the way, he visited the headquarters of the Australian administration, Telecom Australia, and the University of Canterbury in Christchurch, New Zealand. On his way back from the other side of the world, Rekdal visited the Indian Hill Bell Laboratories at Naperville, Illinois, where he gave a presentation of Chill before a large audience.

In Australia and New Zealand, Rekdal met with representatives of the telecommunication administrations, and discussed the experience of using Chill in the Nordic administrations, as well as general aspects of programming expertise in telecommunications. In his travel reports, Rekdal noted that representatives of the Australian telecommunications administration, as well their counterparts in New Zealand, had expressed an interest in software as a means to become more independent of their suppliers, echoing some of the thinking behind the programming language. Representatives of both administrations expressed an interest in the use of Chill to achieve such a position. [90]

At his visit to Bell Laboratories at Naperville, Rekdal encountered what at the time was widely acknowledged as *the* leading research and development organisation in the field of computing as well as telecommunications. Rekdal's mission was to gather information about how the organisation was administered as well as their activities on software development and use. With evident awe, Rekdal noted: "During the summer of 1979, it was decided to erect a new building to house 2000 people all working on various software aspects. The planning took only two months and the building will be finished in the summer of 1980!" [91] Furthermore, he

[89] Kristen Rekdal, "Reiserapport fra CCITT Implementors Forum, CCITT WP XI/3, TELECOMM Australia, University of Canterbury, Bell Labs, 20/9 – 16/10 1979," box "Implementors Forum 9. møte Melbourne, Sept. 1979, Serie O", KRC.
[90] Ibid.
[91] Ibid.

also noted: “50 per cent of the 1600 researchers and engineers at Indian Hill work with software. In the future, about 70-80 per cent will be working on software.”[92] Obviously, he was no less impressed with these numbers.

The Indian Hill establishment of Bell Labs was at the time involved in the development of the software for the Bell’s new switching system, called No. 5 ESS.[93] This involved much novel technology, some of which Rekdal noted. He mentioned the use of mini-computers in place of mainframe IBM computers, all fitted with the Unix operating system, developed at Bell Labs, and the use of the programming language C, also developed in-house, which warranted a particular description in Rekdal’s report: “[C] is a middle level, partly machine dependent, systems implementation language.”[94] At this time, C and Unix were almost unknown quantities to Rekdal, the community in the CCITT and the world at large. In hindsight, C became *the* dominant programming language in systems development during the 1980s and beyond.[95] Without possessing the technical refinements of language such as Chill, C would nevertheless go on to achieve a status almost of a de facto standard, even though it was, according to its creator, “quirky, flawed, and an enormous success”.[96] While C was unknown in 1979, Rekdal and his compatriots would encounter C numerous times; more often than not viewed as a direct competitor of the programming language he had been instrumental in creating. Both in technical and organisational terms, the differences between Chill and C were considerable. C did not strive for a high level of abstraction; it was fairly low level and could consequently produce fairly efficient code. Furthermore, its association with the operating system Unix made the environment of the programming language explicit, something not possible with Chill.

On his travels Rekdal made contacts, gathered information and tried to understand whether the future of Chill was looking promising or grim. As

---

[92] Ibid.

[93] On the history of No. 5 ESS, see Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 379-89.

[94] Kristen Rekdal, “Reiserapport fra CCITT Implementors Forum, CCITT WP XI/3, TELECOMM Australia, University of Canterbury, Bell Labs, 20/9 – 16/10 1979, box “Implementors Forum 9. møte Melbourne, Sept. 1979, Serie O”, KRC.

[95] The history of C has been described by its main creator, Dennis Ritchie, in a paper from the second History of Programming Languages conference. See Dennis M. Ritchie, "The Development of the C Language", in *History of programming languages II*, ed. Thomas J. Bergin and Richard G. Gibson (New York; Reading, Mass.: ACM Press; Addison-Wesley Pub. Co., 1996). An evidence of the tangled nature of programming languages and telecommunications is the fact that Ritchie participated in one of the meetings in CCITT’s Implementors’ Forum. For details about participation, see chapter five in this dissertation.

[96] Ibid.

evident above, two aspects were of particular interest: the role of the telecommunication administrations and the technological choices of the manufacturers – in particular the role of competing programming languages, like C. However, at the time of writing, Rekdal was more concerned about a competing programming language named Ada, which increasingly looked like a viable contender to Chill for programming telecommunication equipment.

## "There need not be a conflict" – Chill meets Ada[97]

From around 1977, Chill had a competitor. The US Department of Defense embarked on standardising a programming language for the creation of real-time and embedded software systems. [98] The language would later be named Ada and attracted a lot of interest and investment, also from European actors. Some of those who became involved with Ada were also involved with Chill, creating conjunctions between the two projects during the period of the Implementors' Forum. In 1979 and 1980, the participants of the Chill project often encountered the Ada project, either in the form of arguments against their own project or through common projects or meeting places. Sometimes the encounters were outright skirmishes, though others were of a more collaborative character. While the Chill designer Kristen Rekdal claimed, "there need not be a conflict" when asked about the relationship between Ada and Chill in 1980, an impression of competition and conflict remained.[99] On the following pages, I am concerned with the meetings and encounters between Chill and Ada in the period when the Implementors' Forum was active, that is, up until 1980.[100] Particularly, I am concerned with how the competition influenced decisions and directions concerning programming language design and implementation. I will return to how Ada influenced the diffusion of Chill in chapter six.

The design process of Ada differed somewhat from that of Chill. Whereas the CCITT did not manage to lure an expert of the standing of Niklaus Wirth to take on the project, the Department of Defense opted for a competition between four contractors to produce a prototype language. By

---

[97] The quote is from Kristen Rekdal, "CHILL, ADA and ESL", RUNIT Notat, 3 March 1980, box "NTT 77-2 NT-Programmspråk 1979 – 1980", KRC.

[98] Whitaker, "Ada—the project: the DoD high order language working group".

[99] Kristen Rekdal, "CHILL, ADA and ESL", RUNIT Notat, 3 March 1980, box "NTT 77-2 NT-Programmspråk 1979 – 1980", KRC.

[100] The following account is based on two types of sources. One is the retrospective article by William A. Whitaker, who chaired the Department of Defense's working group on high-level programming languages. Secondly, I have utilised a number of documents, reports and notes made by or made available to actors involved in the Chill project, many stored in KRC.

May 1979, the department decided to go forward with a proposal from the French company CII Honeywell Bull. The language was developed by the French company under contract to the United States Department of Defense during 1979 and 1980, a somewhat peculiar alliance if one considers France's considerable nationally oriented industrial policy towards telecommunications and computing up until at least the early 1980s and its hostility towards American initiatives in general.[101] That Ada would also attract considerable support from the European Economic Community (EEC) from the 1980s only supports the description of Ada as a somewhat peculiar alliance.

The final reference manual of Ada was delivered in July 1980, and consequently finished just about the same time as the CCITT high-level language was getting its final seal of approval from the CCITT. The project was led by Jean D. Ichbiah, a founding member of the IFIP working group 2.4, a group of computer scientists who under the auspices of UNESCO discussed issues of system programming languages, grew out of the conference on machine-oriented higher-level languages (which was a term disliked by Ichbiah) in Trondheim, Norway in 1974.

Choosing Ichbiah's group as the language designers also meant that a large group of European computer scientists was engaged, both as reviewers of the language proposals on behalf of the DoD, and later on as part of what was to be an emerging Ada community. Even some of the participants in the CCITT project would be engaged on Ada. However, the relationship between Ada and Chill in the early phase of Ada was not one of collegiate spirit. Whitaker, who led the work on behalf of the Department of Defense, has made one comment about this:

> The CCITT developed a common high order language for international use in communications. This was done at the same time as the HOLWG effort, and I was told off-line by members of the developing committee that they made a policy not to communicate with the competition. In any case they never answered any of my letters. CHILL was the product of that development.[102]

Did the Chill camp view Ada as competition? Did they deliberately not communicate with them? The two efforts shared a great deal of common antecedents and meeting points. The IFIP was one, where Remi Bourgonjon met with Jean Ichbiah in the 2.4 working group. Dines Bjørner would, almost at the same time he was about to produce the formal definition of

---

[101] This is an altogether too simplistic argument, although it at least grasps part of the essence. For a brief, but more balanced view, see Philippe Mustar and Philippe Larédo, "Innovation and research policy in France (1980-2000) or the disappearance of the Colbertist state", *Research Policy* 31, no. 1 (2002).

[102] Whitaker, "Ada—the project: the DoD high order language working group".

Chill, embark on a similar project with Ada. So non-communication could hardly have been the case. However, the Chill group certainly felt the pressure.

The Chill group did not give up without a fight: and in 1979, they had a head start: According to Rekdal, it seemed "that CHILL has more practical experience and a wider commitment behind it compared to ADA. CHILL has been through a shakedown period which ADA is still lacking. There is no reason to believe that ADA will pose fewer problems in this respect than CHILL has."[103] Furthermore, the Chill group believed in peaceful co-existence, since the backing from the telecommunications industry was secured:

> There need not be a conflict: To be viable a language needs the support of a sufficiently large user community with homogeneous objectives. Trying to incorporate basically diverging objectives may ultimately erode user support. CHILL is, in this respect, in a fortunate position. The potential use community consists of all members of the CCITT. These members have declared their common objectives by joining the CCITT.[104]

This "fortunate position" was not guaranteed for the future. Chill's supporters feared that the Ada project would deflect attention and support from their programming language, and worried about the consequences: "If support wavers at this time, the current momentum will be lost and the language will die."[105] To the Chill group, the differences between the two projects were both technical and political: the Ada project put the Department of Defense in control of language development. To the CCITT representatives, this fact meant that it served US national interests rather than those of the international community. Chill advocates underlined the contrast: "CHILL on the other hand has been developed by the CCITT which is a truly international body working under the auspices of the United Nations."[106] It would also have been politically impossible for the CCITT to standardise a language over which it had no control.

Nevertheless, Chill's momentum was wavering, and the spotlight was firmly on Ada, although the EEC first considered finding a third way towards programming real-time systems. In 1978, the EEC started investigating the possibility of creating a new European programming

---

[103] Kristen Rekdal, "CHILL, ADA and ESL", RUNIT Notat, 3 March 1980, box "NTT 77-2 NT-Programmspråk 1979 – 1980", KRC.
[104] Ibid.
[105] Ibid.
[106] Ibid.

language, tentatively called the European Systems Language.[107] The German Siemens and French CII Honeywell Bull carried out the study, and concluded that rather than opting for a new programming language, the Ada programming language should be chosen. The European Commission followed this up by granting funds for specific European Ada-related projects during the early 1980s. The EEC actively encouraged the formation of the interest group Ada Europe in late 1979, before the Ada language was even finished and fully defined. At the inception of the Ada Europe group in March 1980, more than 40 individuals or organisations had an active interest in the language. The group drew considerable interest from general computer manufacturers as well as the computer science community. Already at its outset, 16 projects involved either in direct language design activities, feasibility studies or implementation projects were underway in Europe, involving technical universities, research establishments and computer companies.[108] Much of the interest in the language came from the formidable backing of the US Department of Defense, as a prospective standard language in the development of many military systems would mean a huge market. The backing also stemmed from the choice of the French language designers, in particularly Jean Ichbiah, who had a high standing within European computer science circles and the constituency of the IFIP. Furthermore, the language was considered technically advanced. As the initiator of the European group, Brain Wichmann of the UK National Physics Laboratory, stated in his opening remarks to the group's first meeting:

> Even if Ada had nothing special to recommend it, concentrating on one language to avoid the inevitable duplication of effort would be a worthwhile gain. However, the package concept in Ada should allow tailor made systems to be built largely from standard components – components which are pre-compiled and whose is checked for type validity by the compiler.[109]

By 1982, the EEC had actively pursued and sponsored projects related to Ada as a major part of the community's data processing programme. In particular, the sum of 6.4 million ECU was put into two large Ada compiler projects from 1980, along with a number of smaller projects also given

---

[107] For this, I rely on "Community Data-processing Policy", Communication from the Commission to the Council, Brussels, 22. July 1982. Available from Archive of European Integration (AEI), hosted by University of Pittsburgh at http://aei.pitt.edu/

[108] List of identified projects on 4 December 1979, attached to B. A. Wichmann and R. Gilbert, Proposal for a WGS-ADA-Europe activity, 5 December 1979, box "NTT 77-2 NT-Programmspråk 1979 – 1980", KRC.

[109] B. A. Wichmann, "Ada-Europe: Some opening remarks," 1 February 1980, box "NTT 77-2 NT-Programmspråk 1979 – 1980", KRC.

resources by the EEC.[110] The extent of support and investment behind Ada was in another league compared with Chill. Where the Chill project depended on local networks for compiler projects, the Ada project was coordinated and published on the Arpanet, the US DoD-sponsored forerunner of the modern-day Internet.[111]

At the establishment of Ada Europe, Remi Bourgonjon was invited to discuss the possibility of harmonising the two programming languages. Bjarne Däcker, who participated on behalf of L. M. Ericsson at the inaugural meeting of Ada Europe, remarked that the discussions between Bourgonjon and Ichbiah were not particularly constructive: the Ada supporters argued that it would be a waste of resources to create two standardised programming languages for almost the same type of users, and that Chill should be disbanded. The Chill supporters, on the other hand, thought that they had a head start, since they already had some compilers working at this time.[112]

Däcker was impressed by the Ada effort in general, and believed that L. M. Ericsson should reconsider their contribution to Chill. The Swedish firm had participated in both the Team of Specialists and the Implementors' Forum, but the emergence of Ada made Däcker believe that their future projects should be based on that programming language. This was based on two observations. Firstly, that the functionality of the language would be able to match the needs of L. M. Ericsson, if the representatives of L. M. Ericsson were able to influence the final design of the language. Secondly, that the level of backing for the language would ensure that a significant amount of supporting tools and resources would be available from other companies.

---

[110] The European currency unit (ECU) was an artificial currency that was used by the member states of the EEC as their internal accounting unit. It was conceived in 1979. To compute the real value of the Ada support to a present value is complicated, as it would necessitate taking into consideration the rate of price changes in the EEC member countries, and in the end, the conversion into a present-day currency. The composite nature of the ECU makes this conversion difficult. Furthermore, it would also be necessary to take into account that there exist different Later in this thesis, where investments based on a single currency are given I will provide measures in 2010 US dollars, adjusted with the Consumer Price Index. Here, such an adjusted and converted measure is not possible to present.

[111] Janet Abbate, *Inventing the Internet*, Inside technology (Cambridge, Mass: MIT Press, 1999).

[112] This is revealed in a travel report by Bjarne Däcker, which was made available to the Nordic Chill project. See Bjarne Däcker, "Processkommunikation i Ada – resrapporter", 6 March 1980, box "NTT 77-2 NT-Programmspråk 1979 – 1980", KRC.

During late 1979 and 1980, L. M. Ericsson made moves towards influencing the implementation of concurrent language constructs in Ada, and became increasingly involved in the Ada project. If everything fell into place, L. M. Ericsson were willing to drop all other programming language activities, as both their own Plex and Chill were believed to be inferior to the coming Ada language.[113] According to Däcker, L. M. Ericsson's participant in the Ada process, there was one danger in dropping Chill in favour of Ada, and that was related to the possibility of Chill receiving some sort of exclusive monopoly in telecommunications. Däcker called the representatives of L. M. Ericsson in the CCITT to do everything they could to prevent such a monopoly, and also to discuss Ada in the CCITT.[114]

Other industry actors shared L. M. Ericsson's growing scepticism towards Chill, and some of the participating manufacturer active in Chill would soon defect. John C. D. Nissen of the British General Electric Company (GEC) was appointed by the Ada Europe group to be responsible for issues concerning Ada in telecommunications.[115] The French administration felt a strong push towards Ada, as it was strongly related to the French Honeywell-Bull consortium and the language designer Jean Ichbiah. At the executive level of the CCITT, the chairman of the CCITT Study Group XI, which was responsible for the Chill project, held discussions with people in the US Department of Defense on the issue of Chill versus Ada during the spring of 1980.[116] However, none of these efforts would lead to the immediate disintegration of the implementation part of the Chill project, as it was not until later into the 1980s that Ada really started to gain support.

Summing up, during the last months of 1979 and first months of 1980, a number of actors tried to shelve Chill in favour of Ada. However, the CCITT did not change its mind: the CCITT plenary assembly unanimously approved the language in 1980 and it was then formally endorsed as the language to use for telecommunication applications worldwide. Furthermore, the CCITT was not interested in taking part in the harmonisation of other languages. However, the adoption of Chill would be seriously hampered by

---

[113] The arguments was made explicit in Bjarne Däcker, "Processkommunikation i Ada – resrapporter", 6 March 1980, box "NTT 77-2 NT-Programmspråk 1979 – 1980", KRC.

[114] Ibid.

[115] John Nissen, "The current state of ADA tasking and application in communication." 1 February 1980 (Sent to Ada UK Telecomms Subgroup, Ada UK Tasking Subgroup, Ada Europe), box "NTT 77-2 NT-Programmspråk 1979 – 1980". KRC.

[116] J. S. Ryan to P. Sterndorff, 14 May 1980, box "NTT 77-2 NT-Programmspråk 1979 – 1980", KRC.

the continuing discussions along the Chill versus Ada lines. I will return to this in the next chapter.

## Some conclusions

Chill was approved by the CCITT Plenary Assembly in November 1980.[117] The slow-moving ITU had finally got its programming language. To the ITU, Chill was an UFO created in the outer spheres of the CCITT. To many of the participants in the Forum and the Team of Specialists, the ITU was an alien landscape. Nevertheless, the language was finished, much to the surprise of even its creators. Summing up the final part of the Implementors' Forum, Kristen Rekdal made the following remarks:

> The conclusion is that we have achieved what few thought possible five years back: Chill is finished. The definition is better and more thorough than most. 10 – 12 compilers are already in progress or planned. More than a hundred programmers have been educated in Chill. Chill will be used for at least four new, large-scale computer controlled telephone exchanges that will be put into the market in the first half of the 1980s.[118]

Evidently, there were high hopes. However, there were also lots of uncertainties. The emergence of Ada meant that Chill had a competitor, backed by the large and powerful US Department of Defense, a competitor that was considered a real challenge to the life of Chill. According to a comparison between the two languages made in 1981, both languages were adequate for their stated purpose, although with slightly different strengths and weaknesses. Still, as the comparison concluded: "There is considerable evidence that Ada will become a well-supported standard in virtually all embedded computer applications."[119]

This chapter has highlighted how community-level norms and organisational level strategies influenced the fate of Chill up until its final ratification. I have in particular argued that the processes of knowledge codification, where knowledge about software design and telecommunications was spelled out and turned into syntax and semantics, should be understood as a process of alignments of different development virtues: in the case of the implementation of concurrency concepts, the alignment of the ideas of Bourgonjon and Rekdal on the one hand, and the

---

[117] Minutes of the Plenary Meeting, 10 – 21. November 1980, CCITT, ITUA.

[118] Kristen Rekdal, " Reiserapport fra CCITT's Implementors forum, Melbourne 24 – 29 September 1976", box "Implementors Forum 9. møte Melbourne, Sept. 1979, Serie O", KRC. My translation.

[119] R. T. Boute and M. I: Jackson, "A joint evaluation of the programming languages Ada and CHILL", in *Software Engineering for Telecommunciation Switching Systems* (Coventry, United Kingdom: Institution of Electrical Engineers, 1981).

proposals from L.M. Ericsson depended on the intervention of formal computer science. In the case of formal definitions, the practical concerns of the implementation group aligned themselves with what they believed would be the virtues of formal definitions. However, this did not go as well as planned. The formal definition was late on arrival and did not assist the compiler designers to any large extent.

This is not to say that these realignments were independent of business strategies. The sudden interest in concurrency showed by the Ericsson representatives is just one example of how intertwined these considerations were at the time. Ericsson's growing affinity towards Ada further underlines how the rapidly fluctuating business strategies were able to shape the life of Chill beyond the norms of the community of technical practitioners that had dominated its early emergence.

The period of the Implementors' Forum was also a period marked by prospective users and compiler designers. Compilers were made in companies like the ITT, Philips and Siemens. The Nordic administrations sponsored an ambitious compiler project contracted to Runit, which later would evolve into a project involving a number of manufacturing firms. Through these encounters with prospective users, the language designers gained knowledge about how various language concepts would be used. As shown above, not everyone was impressed. The external pressure from prospective users was further fuelled by the debate about the future of Chill in the wake of the Ada project.

As a whole, the work of the Team of Specialists and the Implementors' Forum framed the period of emergence and the initial birth of the programming language Chill, which involved both programming language design and trial implementations through the constructions of compilers. The close-knit community of technical practitioners never dominated the scene fully. Already at the very initiation of Chill, strategies formulated at the organisational level were of great importance. As shown in this chapter, the feedback from the field trials was allowed back into both programming language design and implementation, but to some extent, this also hinged on various strategies made at the organisational level of many a telecommunication manufacturer.

By 1980, it was time for using Chill in real systems and application programming, as the language was about to diffuse beyond its initial set of early adopters. This is the subject of the next two chapters.

# 5. Large organisations and large systems: the use of Chill in large telecommunication manufacturers

During the 1980s, Chill was used by large industrial manufacturers that developed and manufactured large and complex switching systems. The ITT, Siemens and Philips all applied the programming language to develop large-scale public switches. Chill was used when programming switches that were widely used, making one participant of the CCITT project remark: "Since large companies as e.g. Alcatel and Siemens sell their systems all over the world, Chill is passively used by hundreds of millions of people." [1] Eventually, switches developed in Chill would be put into operations in countries as different as Germany, Pakistan, Norway, Indonesia, Brazil and many more along the way. Implicitly, the diffusion was impressive.

This chapter is concerned with the pattern of Chill's use and diffusion among industrial manufacturers of telecommunication equipment. It starts out with three reasonably detailed case studies of the use in the ITT, Philips and Siemens, which each highlight different reasons and patterns of use. In the Philips case, the study also involves the American AT&T, as Philips and the American AT&T formed a European joint venture to produce telecommunication equipment in 1983, named AT&T and Philips Telecommunications.

The ITT's telecommunication division also became a joint venture. It was sold off to the French company Alcatel in the mid-1980s, also initially as a part of a joint venture.[2] I follow the use of Chill in the new combined company of Alcatel and ITT. Following this, I present a study of the use of Chill by the German manufacturer Siemens in their switching system, the EWSD.[3] Rounding off this chapter, a more comprehensive survey of the use and implementations of Chill in the first half of the 1980s is presented. I consider how some of the firms that had taken part in the early phases of the CCITT cooperation defected from the Chill party. Given these mergers and acquisitions, this chapter casts light on how the changing corporate environment and the political economy of telecommunications influenced and shaped the diffusion and use of Chill throughout the 1980s. The lack of use of Chill by L. M. Ericsson, which had participated in the Chill project extensively until 1980, will be more thoroughly analysed in chapter six,

[1] Jürgen F. H. Winkler, "CHILL 2000", *Telektronikk*, no. 4 (2000).
[2] Rand V. Araskog, *The ITT wars* (New York: Holt, 1989); Christensen, "Switching Relations: The rise and fall of the Norwegian telecom industry".: 259 - 266.
[3] EWSD was an acronym for Elektronisches Wählsystem Digital, or in English, Digital Electronic Switching System.

together with the stance of the Swedish telecommunication administration on the use of Chill.

## Programming systems

In chapter four, I reviewed early implementations of Chill and the creation of programs that translated code written in Chill to machine code, the compilers. When concerned with Chill use in this chapter, we are primarily interested in use beyond implementations, meaning the use of Chill when programming telecommunication systems or prototypes of such. This is what often is referred to as *systems programming*.[4] In the particular case of programming telecommunication systems, such programming involved the development of software that would perform a few dedicated functions with real-time computing constraints, on hardware that was tightly integrated or embedded with the software systems. Compared with the more conventional programming of applications that a normal user would interact with, so-called *applications programming,* systems programming was "closer to the machine". Systems programming had more severe hardware constraints than what typically restricts application programming, but at least the limits were known in advance.

Such systems programming was considered very different when compared with application programming: so much so that programmers and their tools tended to be specialised in one or the other. However, the programming of switching systems was considered even more esoteric than normal computer systems programming.[5] So-called time-sharing techniques that had been developed for real-time computers used in scientific or business applications were also considered to have "little relevance to the requirements of electronic switching systems".[6] Consequently, the use of Chill we analyse in this chapter was related to the development of a very peculiar type of systems and not comparable with what is normally considered as systems programming. In some respects, various elements of such telecommunication systems were even considered as a variant of applications programming, like the programming of the important call-

[4] To my knowledge, there is no established and overarching classification scheme for programming languages, but the term systems programming is often used in references and textbooks.

[5] One contemporary and rather exotic source, which describes the Japanese D-10 switching system and its software, is Takamura et al., *Software design for electronic switching systems*.

[6] Ibid., 16.

processing application.[7] Nevertheless, in the following, I will use the term systems programming in relation to the use of Chill, as I consider it closer to what is conventionally understood as systems programming than anything else.

While programming telecommunication switches was characterised by the same limited programming facilities that characterised more conventional systems programming, its uniqueness was related to the fact that it had to be capable of dealing with very large numbers of simultaneous calls and process these calls in parallel. Furthermore, the reliability with which it should be able to deal with these concurrent calls was almost extreme, as the example of two hours of total disruption of service in the life span of 40 years, as pointed out in chapter two. One would have to organise the software accordingly. In the early days of computer-controlled switching systems, the software typically came of two kinds, programs involved in the execution of the switching functions and programs involved in various administrative processes. The programs would typically be provided for by an execution environment, an operating system, holding everything, including data transfer, memory management and timekeeping, in check. The main part of the execution part would be the call-processing program, which controlled the switching operation of the calls originating from telephones or trunk lines and the routing of the calls to the correct receiver. The administrative program would be involved in collecting charging information used for billing, monitoring the grade of service the system could provide, and facilitating the modification of the data stored for particular subscribers or lines, like directory numbers and other features. In addition to these, a number of maintenance programs, which only came into operation if faults had been detected, were necessary.

All in all, the programming involved in the development of a full-scale switching system was extensive and varied. A full-blown switch like the EWSD from Siemens could have 100 million lines of code inside, which would require 10,000 man-years to produce.[8] Chill was intended to be a general programming language that should be able to assist the programming of programs. An important aspect of systems programming of this type was also its managerial aspects, in particular how to make it possible for a large number of programmers, often in the 100s or even 1000s, to work on the

---

[7] One example of this naming convention is Mark W. Clark, Hans-Joachim Hey, and Gerd-Arnold Schlaffke, "EWSD software modularity - smoothing the way for performance increases", in *Global Telecommunications Conference* (Hollywood, Florida: IEEE 1988).

[8] Hans-Eugen Binder, "A telecommunication development: Siemens' digital switching system, EWSD", in *Proceedings of the 18th international conference on Software engineering* (Berlin, Germany: IEEE Computer Society, 1996).

same systems, and in some cases, on the very same code. The large numbers of programmers were necessary to be able to deliver the systems quickly. Consequently, a major undertaking when applying Chill to the development of the above-mentioned systems programs, be they administrative programs or operational ones, was to provide for such large-scale efforts. Between the process of writing the code and handing it over to the compiler, another set of tools can be applied to ease the work associated with software development. This part of the tool chain, including the compiler, has often been called the *software development environment.*[9] In the following, we will often encounter such programming tools that existed as part of a larger software development environment built around the programming language Chill. In chapter six I will further analyse how the experience from creating programming systems fed back into the language design.

## Chill in the ITT

ITT was one of the original collaborators in the Chill project and had been active in both the Team of Specialists and in the Implementors' Forum, although their role in both groups was always somewhat peripheral. The ITT participants were often more observers than active contributors, although we noted the pioneering contributions of the ITT in relation to the concurrency debate in chapter four. Nevertheless, it came as no great surprise that ITT would be one of the first major switching manufacturers that adopted Chill. In fact, already from 1977, ITT was committed to using the programming language, at a point in time the language was only early in its implementation phase.

The early commitment would be put under pressure as the whole organisational framework of the ITT telecommunication operations changed in the mid-1980s, as the ITT's telecommunication division was sold off to the French company Alcatel, initially as a part of a joint venture, but later as a full merger.[10] At that point, the ITT had used Chill extensively when developing their large-scale switching system, System 12.[11] Alcatel would stick with Chill when they continued to produce the System 12. The new

[9] The terminology of software development environments was, to some extent, a thing of the 1980s, where the term proliferated in conference proceedings, like those from the International Conference on Software Engineering, and in journals such as the *Software Engineering Journal*. As such, the term is historically correct when discussing the period under consideration, although it might not be in vogue at the moment. A similar term, so-called programming environments, appeared earlier, in the late 1970s, and implied much of the same thing.

[10] Araskog, *The ITT wars*; Christensen, "Switching Relations: The rise and fall of the Norwegian telecom industry", 259 - 266.

[11] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 415 - 24.

firm would also port part of the software for another Alcatel switching system, the E10, to Chill.[12] Furthermore, during the ITT years and after the Alcatel merger, a number of the company's subsidiaries were involved in their own Chill-based development projects, for example in Austria and in Norway, independent of the System 12 and the E10 projects.

The development of ITT's major switching system, System 12, was programmed in Chill and introduced to the market in the mid-1980s. This involved the development of implementations and systems at a number of places in the USA, UK, Germany and Belgium. The French system E10, first developed by Alcatel and later a part of the portfolio of the outcome of the merger between Alcatel and ITT's telecommunication activities, would gradually use Chill in its software development, after starting out with a number of other programming languages. Initially, the E10 was a result of an earlier merger between the civilian telecommunication division of Thomson and CIT, owner of Alcatel, which were put together just before the joint venture between Alcatel and ITT.[13] The local initiatives to use Chill were going on within national subsidiaries, most notably in Norway and in Austria. The development of the so-called nodal switch system, a military switching system, in the Norwegian ITT subsidiary STK was done in Chill, a project that would later spur the development of a fairly successful private branch exchange called Digimat.[14] STK had been active in the Nordic Chill implementation projects from the late 1970s, and was an early adaptor of the Nordic Chill compilers in many of their development projects. In the following, I will dwell on the reasons to use Chill in all these three settings, within the System 12 project, within the Alcatel merger and in the Norwegian subsidiary STK.

The history of the System 12 switch is rather vague. The authoritative text on everything about digital switching, Chapuis and Joel's second volume of *100 years of Telephone Switching*, has even reverted to the narrative structure of a "fairy tale" to sketch the upbringing of the System 12.[15] The authors make it clear that there are many different versions of the history of the System 12, versions that involve technical wrangling and financial difficulties within and between various parts of the ITT. Eventually, both the financial problems and the technical difficulties led to the sale of ITT's telecommunication divisions to Alcatel in 1987. However, one thing stands out in the Chapuis and Joel account, regardless of its vague

[12] Ibid., 319-31.
[13] To be precise, Thomson-CSF Téléphone was taken over by "Compagnie Industrielle des Télécommunciations" (CIT) in 1983.
[14] Christensen, "Switching Relations: The rise and fall of the Norwegian telecom industry", 175-205.
[15] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*.

and "fairy tale" structure: the plethora of ITT subsidiaries involved in the System 12 development, and in particular how this was the case with its software development. This distributed and international mode of software development has also been confirmed in conversations with software developers associated with the project.[16] In the early 1980s, 10 locations in different countries plus the USA were concerned with the development and/or the supply of switching systems in the ITT and about 800 programmers were involved in the development, spread across a large number of these locations.[17] As described in chapter four, the early software developments related to the System 12 project were initiated from two of the ITT's central research establishments in the USA, the ITT Programming Technology Center in Stratford, Connecticut, and the Advanced Technology Center in Shelton, Connecticut. Over time, the programming migrated from the USA to Europe. The Americans kept the development of the operating system of the switch, further tools were developed by the subsidiary in the UK, the database of the switch was worked on in Germany while all software maintenance was handed over to the BTM company in Belgium. Subsidiaries in other European countries developed the call handling routines for the systems deployed in their own country.[18] By that time, the numbers of people engaged in software development related to System 12 had grown far beyond the initial 800.[19]

ITT committed itself early to the use of the Chill programming language, mainly because it was in a hurry. Only Siemens and Philips embarked on similar "early adopter" projects, all three trying to bring out a new generation of digital switches in the early 1980s. The decision to put their weight behind Chill was also about striking a balance: the status of an international standard was an asset to the multinational company that wanted to coordinate development efforts across multiple local subsidiaries. It was, on the other hand, also a possibility to adapt it to the needs of the local ITT companies, like the possibility to accommodate it to a variety of target machines and building tools that would be portable between host machines. Furthermore, when the decision was made in 1977, the Chill definition was still in a state of flux, lacking in several respects features that would be part of the finished recommendation in 1980. Consequently, this created an

[16] Neil Olsen, Tom Love and Capers Jones in various emails to the author, February 2011.
[17] R. W. Daley and T. A. Haque, "ITT 1240 Digital Exchange – CHILL programming Environment", in *Second CHILL Conference* (Lisle, Illinois1983).
[18] Neil Olsen, in various emails to the author, February 2011.
[19] Neil Olsen, who moved from being a junior software developer to become a software architect at the ITT in this period, estimated the number of programmers to be about 2000 at its high point.

understanding of an apparent flexibility of the standard: the ITT could fill in the blanks themselves, as well as make restrictions of it into their own subset.

Another reason for this early backing of the CCITT programming language was also the lack of a future for the existing high-level programming language used by the ITT, the ESPL-1, which was a PL/1-like language created within the company.[20] Summing up, local conditions at the different ITT development and supply sites combined with the lack of technological unity around the existing programming language ESPL/1 created the opportunity to adapt to the CCITT recommendation very early on. The ITT also believed that the technological choice to use Chill would be a valuable asset in the varied national markets they targeted with their new switching system, assuming the standard to be enforced as a requirement for bidding for new tenders.

In use, Chill had considerable shortcomings. Within the ITT, this led to the development of several improvements and local modifications. Two issues proved particular problematic, as explained below by Wen from the ITT Technology Center in Shelton, Connecticut:

> ITT 1240 is made up of different software modules to enable separate compilation. These different software modules have to communicate with each other, or at least, the interfaces across these software modules have to be well defined so that there will be no gaps between these modules. However, CHILL as a programming language does not address these interface mechanism problems. In addition, CHILL does not address the problem of configuration management which is a crucial in a system as big as ITT 1240. This is especially critical since ITT 1240 is developed across multiple design centers around the world.[21]

Consequently, the ITT would put considerable weight behind efforts to improve their toolkits to remedy these shortcomings, leading to the development of their own environments, capable of handling the division of labour they needed. Consequently, ITT's use of the programming language was never "pure." To the ITT, it simply was not a "high enough" language.[22] As a consequence, ITT developed a series of extensions and so-called problem-oriented languages on top of it.

---

[20] Hills and Kano, *Programming electronic switching systems - real-time aspects and their language implications*, 143-46.
[21] W. Wen, "Problem Oriented Languages", in *Second CHILL Conference* (Lisle, Illinois1983).
[22] Neil Olsen, in various emails to the author, February 2011.

The ITT embarked on the construction of a new Chill compiler, the so-called Chill2, from late 1981.[23] As was the case with the ITT's first Chill compiler, this involved the use of outside contractors. This time around, the firm Intermetrics, a firm from Cambridge, Massachusetts, was the contractor, although some of the work was also retained within the ITT. The maintenance of this compiler was eventually moved to the UK, to the ITT Programming Support Centre in Harlow, Sussex. By the early 1990s, however, the Chill compiler was maintained, developed and supported by an outside company, Richard Daley Associates, and their improvements made to the Chill tools used by the ITT represented one of the few continued efforts in Chill tool developments in the 1990s.

The use of Chill in the development of the Alcatel-developed switch E10 was an afterthought. The software in E10 was originally coded in three different programming languages. The call handling was programmed in machine code, the operation and maintenance software in PL/1-like language CPL/1 and the signalling system was coded in PL/M, a language originally created as the implementation language for the operating system CP/M. After Alcatel incorporated the ITT telecommunication operations, it was decided that the next E10 switch should use the same software as the old version, but that the code should be in Chill – the programming language favoured by the newly acquired ITT. To achieve this, Alcatel decided to port the existing software on the E10 switch – and not develop new software. Since the new generation of E10 switches, cryptically named OCB 283, would be based on the Motorola 68020 processor, a completely new set of compilers and tools would be developed. At the same time, tools that would try to automatically translate the code in CPL/1 and PL/M were developed. In the end, the software porting project would involve the development of new compilers, translation tools and manual translation. All together, Alcatel would use 83 man-months to port all the software to Chill, with an additional 66 man-months used to port the machine code used in the call handling to a usable machine code for the new target processor.[24]

The effort towards integration and concentration on Chill within the newly formed Alcatel system had various technical reasons. The mixture of machine code, PL/M and CPL/1-developed parts in the original E10 design were a result of a similar mixture of hardware, using three different processor families within the same switch. The new E10 design was based on a single processor, the Motorola 68020. Consequently, this change created an opportunity to focus on a single programming language (and a single compiler, respectively).

---

[23] The following section is based on Richard Daley, emails to author, April 2011.

[24] F. Hamonno et al., "Switching System Software Base Portage to Chill", in *Fifth CHILL Conference*, ed. Antonio Palma (Rio de Janeiro, Brazil1990).

In the case of the Norwegian military switching system, the adoption of Chill as the programming language had a number of different causes than those discussed above. At the same time as ITT decided to use Chill internationally in 1977, the Norwegian subsidiary STK made the same choice, but within a rather different context: In 1973 STK got a contract from the Norwegian Defence Research Establishment (NDRE, FFI in Norwegian) to develop a small digital switch called the nodal switch.[25] This project was a local initiative based on a number of local relations, between STK and the NDRE and between STK and the research establishment within the Norwegian telecommunication administration. Today, the nodal switch forms one of the technological and commercial bases to the Norwegian subsidiary of the defence company Thales, and during the 1970s and 1980s it was an important part of the operations of STK.[26]

From 1977, it was decided that the software part of this project should be developed in Chill. Here, the fear of an enforced standard could not have been imperative, since the market for the nodal switch was not that of public switching and telecommunication administrations, but that of the military. National relations were important, as the close relations between Runit in Trondheim and STK in Oslo paved the way for a close collaboration on software development within the nodal switch project. The compiler created by Runit within the Nordic Chill project was adopted and sold to STK, which bypassed the internationally coordinated compiler project within its parent, the ITT. STK was, in fact, the first real user of Runit's compiler and the first customer of the soon-to-be-spun-off company Urd.[27] As such, the decision to use Chill within STK was influenced by the nationally oriented relationships of the firm. The technical difficulties were, however, fairly similar to those that were experienced within the System 12 project internationally: the adoption of Chill at a very early point in time meant several of its features had still not been decided, and were not available to the Norwegian implementers. Still, STK relied on the relationship with Runit rather than with their parent company when developing the software and the tools.

Two paths within the ITT, and, from 1987, a third path within the newly formed Alcatel organisation, led to Chill. Within the ITT, this included both the multinational use of Chill within the System 12 project and the local development project of the Norwegian Nodal switch. These two paths were continued within the Alcatel system, after the French

[25] The following is based on Christensen, "Switching Relations: The rise and fall of the Norwegian telecom industry", 180-85.
[26] Ibid., 175.
[27] Stein Erik Ellevseth, "The SDS Software system", in *Third CHILL Conference* (Cambridge University1984).

procurement of the ITT's telecommunication assets in 1987. This acquisition led towards the third path of Chill use, one concerned with porting the existing code of the E10 switch to Chill and the development of new E10 software in this programming language.

## Encountering C – Chill, Philips and the AT&T

Philips was one of the main participants in the creation of Chill. Their young programmer Remi Bourgonjon had led the work in both the Team of Specialists and the Implementors' Forum. His successor, Kees Smedema, continued the strong presence of Philips within the CCITT in the study period right after the standard was made an official CCITT recommendation, from 1981 to 1984.[28] Smedema was also actively promoting the use of Chill within the telecommunication division of Philips. One of the largest undertakings of Chill compiler design was begun; a project that continued well beyond early implementation experiments and was put into real use in the early 1980s. However, Philips joined the AT&T in a European joint venture from late 1983, radically altering the possibilities to use Chill in the development of new switching systems – and more generally, radically altering the possibilities to continue the development of a Philips-led switching project altogether.[29]

The joint venture meant a complete halt in the development of the Philips telecommunication systems – as the strategy of the venture was to use the Philips organisation to market the AT&T-developed No. 5 ESS system in Europe. As such, the whole impetus of the Philips involvement in the Chill project seemed out of place. Some of the development and production of AT&T's ESS system was continued in Europe, and in particular in its development laboratory in Brussels, Belgium, the Programming Languages Support Group. One of the efforts was to develop a European or international version of the software of AT&T's switching system, a version based on Chill. The joint venture and the decision to create an internationalised software version for the AT&T's switching system created an opportunity to muscle out the credentials of the C programming

[28] Information about the two projects has been obtained through two extensive interviews: Remi Bourgonjon, interview with author, 16 January 2009, Heeze, the Netherlands; Kees Smedema, interview with author, 20 January 2010, Heeze, the Netherlands.

[29] The joint venture has been discussed in Farok J. Contractor and Peter Lorange, *Cooperative strategies in international business* (Lexington, Mass.: Lexington Books, 1988). The following chapter is particularly informative: Karen J. Hladik, "R&D and International Joint Ventures", in *Cooperative strategies in international business*, ed. Farok J. Contractor and Peter Lorange (Lexington, Mass.: Lexington Books, 1988).

language, used in the No. 5 ESS favoured by the AT&T, and the Chill programming language.[30] However, the joint venture would soon crumble. In 1990, Philips withdrew from the public telecommunications market all together, exiting the joint venture with AT&T.[31] As such, the wrestling match was a short one. However, it was still an important moment in the diffusion of Chill.

To understand the skirmish and the outcome of the clash of programming languages within the joint venture, we need to understand a bit more about what went on within Philips before the joint venture, as well as the activities within the joint venture between Philips and the AT&T. Regarding programming languages, the practices within Philips in the 1970s were as diverse as those outside Philips. A plethora of programming languages and low-level coding practices flourished in the various product divisions involved in computer use and development at Philips. While Remi Bourgonjon had envisioned his participation in the CCITT as way to change the low-level coding of assembly-like languages to a more high-level language in the telecommunication division, spurred on by working long hours with the troublesome code in his initial job as a programmer, such a shift could also be fulfilled by applying another existing programming language, a policy favoured by other product divisions within Philips. The computer industry division of Philips were also heavy users of programming languages. PL/1 and Fortran were popular at the time. In the middle of this,

[30] The general understanding of the telecommunications division of Philips has benefited from Herman Oosterwijk, "Switching Technology through Five Decades: Dutch Telecommunications under Change", in *Buidling bridges between ideas and markets*, ed. Frans van Waarden, *Report to the European Commission* (2002). See also Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 240-42. On the relationship between the research department and the product divisions, the detailed history of Philips' Natuurkundig Laboratorium has provided invaluable insight. See Marc de Vries, *80 years of research at the Philips Natuurkundig Laboratorium (1914-1994): the role of the Nat.Lab. at Philips*, ed. Kees Boersma (Amsterdam: Pallas Publications, 2005). On the development of Philips Computer Industry, see Jan van den Ende, Nachoem Wijnberg, and Albert Meijer, "The Influence of Dutch and EU Government Policies on Philips' Information technology Product Strategy", in *Information Technology Policy: An International History*, ed. Richard Coopey (Oxford: Oxford Unievrsity Press, 2004).

[31] AT&T itself spun off its technology company, composed of the remains of Western Electric and Bell Labs, on 30 September 1996, into what became Lucent. On Western Electric and Bell Labs, see Stephen B. Adams and Orville R. Butler, *Manufacturing the future : a history of Western Electric* (Cambridge ; New York: Cambridge University Press, 1999); Kenneth Lipartito, "Rethinking the invention factory: Bell Laboratories in Perspective", in *The Challange of Remaining Innovative*, ed. Sally H. Clarke, Naomi R. Lamoreaux, and Steven W. Usselman (Stanford, California: Stanford Business Books, 2009).

Philips Research Laboratory was actively promoting the use of more modern (when compared with Fortran) and less complex (when compared with PL/1) programming languages, like Pascal or Modula.

On the one hand, the scientific credentials were favoured by the Research Laboratory. On the other, industry favourites like PL/1 and Fortran were held in high esteem in the computer industry division. Between these two, the bastard child that was to become Chill would eventually become the choice of the telecommunication division.

Initially, Kees Smedema was a researcher at the research laboratory, and not that keen on Chill. The research lab in which Smedema was working tried to persuade the different product divisions to switch to what Smedema described as "a decent language",[32] more or less hoping for the adoption of Pascal or Modula as the internal standard. This effort was directed towards the computer industry division and the telecommunication industry division. In January 1979 Philips installed a "Committee on Pascal-like languages", which was usually known as the Pascal Group. The group held a mandate to advise on the use of different programming languages within various product divisions, and the setup of an in-house educational facility in programming.[33] This committee grew partly out of the active advocacy of the research laboratory, and consequently the group continued its effort to persuade the product divisions to switch to Pascal and Modula. The committee represented an important boundary-spanning unit, as it transgressed the corporate research laboratory and the product divisions, involving people like Remi Bourgonjon. After a while, Chill was included among the "favoured languages" of the Pascal Group, and from 1980, Kees Smedema, one of the central members of the Pascal Group, joined Bourgonjon at the Telecommunications division. The strong belief in the virtues embodied in Pascal and Modula had to be replaced by the much more unruly Chill. To the colleagues of Smedema in the research laboratory, the move to telecommunications was likened to the giving up of beautiful women for an old hag. See the figure on the next page for this.

[32] Kees Smedema, interview with author, 20 January 2010, Heeze, the Netherlands.

[33] The committee is described in Smedema, Medema, and Boasson, *The programming languages : Pascal, Modula, CHILL, and Ada*. The committee also published a quarterly newsletter named Compas (Communication on Pascal-like computer languages). Copies of this internal publication have been loaned from the private collection of Kees Smedema (hereafter cited as KSC).

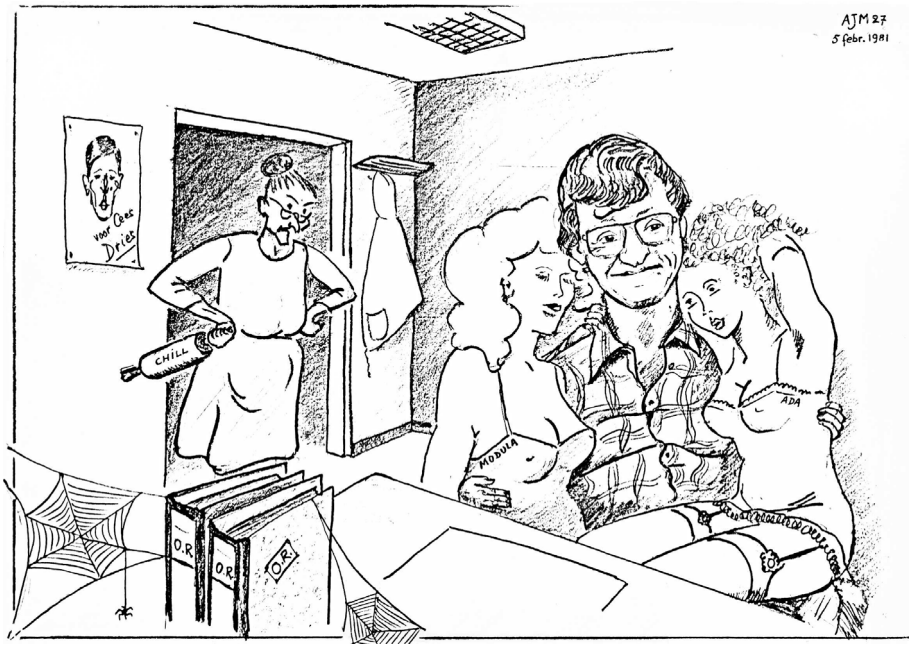


**Figure 5.1 From lightly dressed temptations in research to the strict realities of telecommunications.**[34]

Despite the turning up of noses at the research laboratory over Chill, the programming language was used in the development of two new switches that were to be introduced during the mid-1980s: a small local exchange called TCP 16 and a large trunk exchange called TCP 36. Both were developed from around 1979.[35] The TCP 16 was forcefully stopped fairly

[34] This cartoon was drawn when Kees Smedema left the Philips Research Lab for Philips Telecommunications Systems. It depicts his room in the Laboratory, and illustrates his relationship with the two programming languages Modula and Ada, while his future looms in the background as a strict aunt with the rolling pin of Chill. The books with O.R. referred to Smedema's activities as chairman of the employee representative council. The picture on the wall (voor Cees (wrong spelling) Dries) refers to the Dutch Prime Minister Dries van Agt, which was born in the town in which Smedema lived. Smedema was at the time a member of the Town Council, which wanted to make van Agt an Honorary Citizen, a proposition Smedema opposed. The cartoon was drawn by Alan Martin, a member of the so-called Tuesday Afternoon Club, led by the luminary computer scientist Edsger Dijkstra.

[35] Some details about the software aspects of the TCP 16 are given in D. Hammer, FG. Franken, and P. C. Green, "A distributed operating system for the TCP16 system", in *Fifth International Conference on SOftware Engineering for Telecommunciation Switching Systems* (Lund, Sweden: Institution of Electrical Engineers, 1983).

early in its development, while the latter was continued up until a field trial in the summer of 1983.[36] The early commitment did not pay off, as the joint venture with AT&T changed the playing field completely, stopping all future development of Philips switches, and consequently putting the lid on the TCP 36 project as well.

Despite the joint venture with AT&T, the software developers of the Philips switches did not give up on Chill. This hinged on a different kind of argument than that of using a "decent language", which had led to the adaptation of Chill within Philips in the first place.[37] The crossroads that were the AT&T Philips joint venture opened up the possibility of using Chill in the development of AT&T's new line of switches, the No. 5 Electronic Switching System (5ESS for short). This involved a close encounter with a programming language that was about to make a big stir in the world of both computing and telecommunications: C.[38]

AT&T had during the 1970s developed the programming language C, which they used in all their development projects at the time of the merger. To the developers in Philips, C spelt trouble, as it threatened to replace Chill completely. However, it also created an opportunity to continue their Chill development. As Kees Smedema recalled:

> We tried to use every argument in order to persuade AT&T that they should do different in a European context than in an American context: C is American - Chill is international. You will be confronted with telecommunication administrations which will force you to use Chill. So you better make sure to have Chill in your switches, because otherwise you will be excluded from tenders. Of course we also used technical arguments: C was definitely a lower-level language and less reliable than Chill.[39]

This was, apparently, a sufficient argument, as the joint venture would support and pay for the use and development of Chill-related tools for a few years. The AT&T had to enter the emergent European market because of its loosening grip on the American market due to liberalisation. To get a foot in this market would require humility regarding what were believed to be the future specifications of their new customers, European telecommunication administrations. As a result, the AT&T Philips Telecommunications tried to create a bridge between C and Chill at their Programming Language Support Group in Brussels, both in technical and organisational terms. In organisational terms, the project involved people on both sides of the

---

[36] Remi Bourgonjon, interview with author, 16 January 2009, Heeze, The Netherlands.
[37] Kees Smedema, interview with author, 20 January 2010, Heeze, The Netherlands.
[38] Ritchie, "The Development of the C Language".
[39] Kees Smedema, interview with author, 20 January 2010, Heeze, The Netherlands.

Atlantic, like the designated group in Brussels (an AT&T Philips Telecommunications operation) and at Bell Labs Software Development Systems Department in Naperville.[40] A group of 15 people on both sides of the Atlantic worked on a technical solution to bridge the two programming languages, C and Chill.

Technically, this resulted in a set of tools that facilitated the translation of C code to Chill, and the integration of Chill-programmed code into the 5ESS system. [41] The result was an integrated development environment for both Chill and C programming, including a continuation of the compiler implementation projects that were already underway before the joint venture, although with a different set of target machines.[42] This was achieved without hampering the capabilities of Chill, nor altering the components in the 5ESS development environment. This latter part was the biggest challenge, according to Tom Hornbach of AT&T in Indian Hills: "One thing we discovered is that interfacing two high-level language isn't necessary very difficult. The challenges arise when you take into account a pre-existing software environment."[43] At the time, it was believed that the way Chill was integrated into the C environment of the AT&T provided a model for others interested in implementing transparent interfaces between C and other high-level languages, like Ada and Modula.[44]

[40] "CCITT Languages Shape Products, Development", AT&T Technical Report, May 1986, KSC.
[41] Mary J. Rowe, "Interfacing Chill with existing C-based systems", in *Third CHILL Conference* (Cambridge University: ITT Europe, 1984).
[42] "Chill compiler released on Unix/370", 5ESS Export News Flash, 26 February 1985, KSC.
[43] Ibid.
[44] Ibid.

**Figure 5.2 The bridge that almost was.** [45]

Mary Jo Rowe of Bell Labs summarised parts of the project and its future as follows:

> This work has shown the compatibility and the ease of interfacing the C and CHILL languages. Moreover, it demonstrates that an evolutionary approach is possible to provide a multi-lingual software environment that will allow a graceful introduction of CHILL into current and future C-based software systems.[46]

In 1987, only two years after the full release of a compiler for AT&T's Unix system, the Chill programming environment and the Chill compiler were ended. [47] The legacy of the Chill environment in Philips was summarised in an internal note circulated from Hilversum in 1987: "The CHILL programming environment provided a sound and healthy, though resource intensive method of developing software. It will go into history as a major positive contribution to quality software development."[48] Technically, its legacy was a mixed blessing. Organisationally, the project was one of many signs of a crumbling partnership: the AT&T Philips joint venture soon

[45] Unix/C and Chill was about to be bridged in AT&T Philips Telecommunications. This picture was a poster made for the AT&T Philips joint venture.
[46] Rowe, "Interfacing Chill with existing C-based systems".
[47] Erik Helbo, untitled note, Hilversum, KSC.
[48] Ibid.

disintegrated and Philips would leave the venture and telecommunications altogether. For the people involved in Chill and software development for telecommunications systems in Philips, this meant reallocations and new pastures: Smedema and Bourgonjon moved to the centre for software technology at Philips, set up at corporate level and led by Remi Bourgonjon. Here, work on software improvement techniques was put into use and developed; highlighting the move away from the belief in particular technologies like a programming language as a sole driver of improvements. AT&T would, on the other hand, continue its reliance on C.

The failed partnership marked the end of the telecommunications division of Philips, a transition towards digital switching and computerised systems. At the outset, the company had sought partnership within the CCITT to help ease the transformation. In the end, another partnership, with an American technological leader, would mark its end. On a technological level, this also marked the end of the strong presence of Dutch engineers and computer scientists within the Chill community. Kees Smedema ended his involvement with the core Chill community when the new study period started in 1985. Another indication of this was the total absence of Dutch participants at the fifth Chill conference in 1990.[49] However, some of its legacy continued within the company, as the importance and centrality of programming and software development in other product divisions would only increase during the 1980s and 1990s. In this way, the arguments favoured by those wishing to orient the software engineering practice in the direction of a mathematically oriented computer science would continue to exert influence of the coding practices within Philips.

[49] Apparently, the use of Chill continued in Philips Kommunikations Industrie AG, a part of Deutsche Philips GmbH. At the fifth conference, one paper by authors affiliated with this subsidiary was presented. See A. Bergmann, T. Letschert, and A. Lingen, "CHILL/tss – a System Development Environment for Telephone Switching Systems", in *Fifth CHILL Conference* (Rio de Janeiro: Telebras, 1990).

## Early adopters and evolutions at Siemens

One of the largest systems developed in Chill was created by Siemens, with their switching system EWSD.[50] The first EWSD switch was put into service in November 1980 in South Africa, and by 1982 the system was installed in eight other countries.[51] Throughout the 1980s and 1990s, the system was widely exported, making the EWSD one of the most widely adopted switching systems, together with ITT/Alcatel's System 12.[52]

The use of Chill coincided with an extensive effort to expand into new markets by Siemens. Most notably, it coincided with Siemens opening a development facility in Boco Raton, Florida, in 1979, targeting the so-called Bell Operating Companies, an effort that increased after the AT&T divestiture in 1984.[53] The development of the EWSD switching software was spread around various locations in Europe and the USA.

The first version of the EWSD system was already at the market at the very same moment that the CCITT ratified the Chill proposal as an official recommendation, making use of Chill as the main programming language for the programming of the switching system's central processor. This early adoption was in the similar mode as Philips, the ITT and the NTT, which all committed to the use of Chill in the period of the Implementors' Forum. Just as with these other early adopters, Siemens had been an active participant in and contributor to the work within the CCITT, in particular through Heiko Sorgenfrei, who held prominent positions in both the Team of Specialists and the Implementors' Forum.

Throughout the 1980s and 1990s, Siemens continued to be committed to Chill.[54] By the mid-1990s, it was estimated that Siemens had spent 25,000 staff-years on software engineering at around 20 development centres around the world to develop the software. More than one billion gross lines of code

---

[50] On the EWSD in general, see Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 409-14.On EWSD and its software, see Dietrich Botsch and Hans Eberding, "EWSD, A Real-Time Communication System with High-Level Language Software", *IEEE Transactions on Communications* 30, no. 6 (1982).

[51] On the sale to South Africa, see David Kaplan, "State Policy and Technological Change-The Development of the South African Telecommunications Industry", *Journal of Southern African Studies* 15, no. 4 (1989).

[52] By 1988, the EWSD system had been sold to 32 countries, to a total of 80 telecommunication agencies and had 8 million lines in service. Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 413.

[53] Ibid.

[54] Clark, Hey, and Schlaffke, "EWSD software modularity - smoothing the way for performance increases".

were, by that time, contained in the EWSD Configuration Management Database, a major proportion of this being Chill code.[55]

The use of Chill at Siemens was extensive, probably more so than at any other manufacturer. It was used in the development of the switching execution programs, in the administrative applications and in the development of a number of support and software development tools. This latter category, where Siemens chose to develop support software such as library routines, linkers and testing aids all in Chill, even though they were tools used when developing software rather than tools used in telecommunication systems, was novel and considerably more extensive than in any of the other examples of use.[56] This was no small feat. According to Erwin Reithmaier, a Siemens engineer, it paid testimony to the generality of the CCITT language:

> We think there is no better proof for the general applicability of CHILL […] than the EWSD support system executing now for more than three years under most severe development and mass production conditions.[57]

According to reports in technical journals and at conferences, the experiences were generally positive. One example was a paper presented in the American IEEE's journal *Transactions on Communications* in 1982: "The overall experience of using Chill in a telephone switching system is very positive."[58] In particular, this was related to the effect the programming language had on eliminating errors in the software:

> The main advantage was found in the number of software errors still present in a large software package. A considerable reduction, on the order of one magnitude compared to assembly language programming, was experienced at the point of time when developers, after module and functional test, hand over their product to the integration team for final integration and stabilization. On average, only two errors per 100 lines of code have been found by integration teams. The number of latent faults in a system, which is an indication of stability and failsafe operation, can therefore be estimated to be remarkably small.[59]

Apart from the better error rate provided by the application of Chill, the head

---

[55] Numbers from Binder, "A telecommunication development: Siemens' digital switching system, EWSD".
[56] Botsch and Eberding, "EWSD, A Real-Time Communication System with High-Level Language Software": 1341.
[57] Erwin Reithmaier, "Compilation Control in a Large CHILL Application", in *Second CHILL Conference* (Lisle, Illinois: Bell Laboratories, 1983).
[58] Botsch and Eberding, "EWSD, A Real-Time Communication System with High-Level Language Software": 1337.
[59] Ibid.: 1342.

of the EWSD project and its chief software developer, Dietrich Botsch and Hans Eberding, emphasised two important aspects of Chill: the portability of the code that made it possible to provide software for different switching processors without extensive recoding, and the concurrency concepts built into Chill. As put forward in chapters three and four, these two were important design criteria when the Chill project was started. In terms of concurrency, it was also a hotly debated subject within the Implementors' Forum. As the EWSD project evolved throughout the 1980s and 1990s, Siemens would increase their stake in the Chill developments within the CCITT. Although the Siemens role in the Team of Specialists and in the Implementors' Forum had been of some importance, they were not the most significant of contributors. Later on, and in particular in the latter half of the 1980s and in the early 1990s, Siemens participants spearheaded several important language design improvements in the CCITT.[60] I will return to these advances in chapter six.

## Taking on the World

To account for the diffusion of Chill beyond the efforts mentioned above, a general survey of its diffusion has been conducted. In the following, I account for both the implementations and the system programming efforts that emerged after Chill was ratified by the CCITT in 1980. In 1984, more than 25 different organisations had implanted or were developing compilers for Chill. A much smaller number were developing telecommunication systems with it in terms of programs and applications deployed in telecommunication equipment. The organisations ranged from small research establishments to large telecommunication manufacturers. In general, a large number of compilers were created, while the numbers of actual systems created with Chill were fewer.

Only ITT (through various subsidiaries) and Siemens used Chill in the production of software for switching systems that were put into full operation on a global scale. Philips had, up until the creation of a joint venture with AT&T in the end of 1983, used Chill in the creation of two new switching systems, but the projects were stopped as the joint venture was put into action.[61]

Furthermore, the NTT of Japan used Chill in various ways in the creation of a number of switching systems. From 1979, the NTT had used Chill when developing the D10 switch, which was later followed up in a

[60] Winkler, "CHILL 2000".

[61] Information about the two projects has been obtained through two extensive interviews: Remi Bourgonjon, interview with author, 16 January 2009, Heeze, the Netherlands; Kees Smedema, interview with author, 20 January 2010, Heeze, the Netherlands.

series of other switches.[62] The international telephone carrier of Japan, the Kokusai Denshin Denwa (KDD), also applied Chill in switching projects in cooperation with NEC.[63] NTT, NEC, Hitachi, Fujitsu and Oki Electric were all cooperating in producing the D10/D50/D60/D70 range of switching systems programmed in Chill, using a compiler developed by the NTT.

Chill was first and foremost used and implemented by organisations that had participated in the initial design process: ITT, Siemens and Philips had all been important contributors in the Implementors' Forum and were continuing their backing of the programming language in the early 1980s. However, the language also spread to a number of new markets and countries like Korea, Brazil, China and India, which all embarked on their own switching systems development projects in the 1980s. The Swedish firm L. M. Ericsson, which had participated in both the Team of Specialists and the Implementors' Forum, is the most notable exception to this. By the early 1980s, L. M. Ericsson opted to not use Chill at all, continuing their use of their own programming language Plex and applying the competing Ada programming language in smaller projects, which is something I will return to in the next chapter.

Below, I have compiled an overview of the use of Chill from 1980 onwards. For the period up to 1984, it draws on official reports from the CCITT. For the later years I have added projects that have been reported on at the Chill conferences. Consequently, the survey is most complete for those first four years, while not that all-inclusive for the period until 1990.
I make the distinction between *implementations* and *systems*: an organisation involved in an implementation would typically be working on the creation of a compiler, a linker and various other tools that would assist the development of applications written in Chill. For reasons of completeness, some of the projects that are discussed in more detail in the next two chapters are included in the survey below. In particular, this is the case of use of Chill within independent tools suppliers, which is the subject of chapter seven, and the use of Chill within telecommunication administrations, which is the subject of chapter six.

---

[62] Maruyama, Sato, and Konishi, "NTT CHILL implementation aspects and its application experience". See also Lars-Göran Larsson, "Future Telecommunications in Japan - Policy and Technology", in *Utlands rapport från Sveriges Tekniska Attachéer* (Stockholm: Sveriges Tekniska Attachéer, 1984).
[63] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 426.

| Country | Organisation | Implem. | System | Switching System |
|---|---|---|---|---|
| USA | ITT/ATC | X | X | System 12 |
| | CTE | X | | |
| Austria | ITT Austria | X | X | 5200 BCS (Amanda) |
| | Siemens Austria | X | | EWSD |
| Brazil | Telebras / Embratel | X | X | Tropico RA, EX |
| Poland | Technical University of Warsaw | X | | |
| Japan | KDD | X | X | XE10, XE20 |
| | NTT / Hitachi, Fujitsu, NEC and Oki | X | X | D10 ESS, D50, D60, D70 |
| | Fujitsu | X | X | Fetex 2000/3000 |
| China | Nanjing Institute of Communication Engineering | X | X | PXAJ-500/2000 |
| | 10th Research Institute of China | | X | PXAJ-500/2000 |
| Romania | ITCI | X | | |
| South Korea | ETRI, Daewoo, Samsung, LG, Hanwha | X | X | TDX-1, TDX10 |
| Portugal | INESC | X | | |
| India | Tata | X | | |
| | C-Dot | X | X | DSS (M680X0-based switches) |
| Italy | Telettra | X | | |
| | SIP-CSELT | X | | |
| | Italtel-SIT | X | X | UT-100, LINEA UT |
| | FACE (ITT) | X | X | System 12 |
| Germany | Siemens | X | X | EWSD, ETS, EMS, AIGFON, BIGFON |
| | Tekade (Philips) | X | | |

| | | | | |
|---|---|---|---|---|
| | Standard Electric L. (ITT/SEL) | X | X | System 12, 5600 BCS |
| Norway | STK (ITT) | X | X | 5500 BCS, Digimat 2000, Tadkom |
| | Norwegian Telecom Administration | X | | |
| | Runit / Urd / Kvatro | X | | |
| USSR | Moscow Institute of Telecommunication | X | | |
| | Moscow State University | X | | |
| Netherlands | Administration | X | X | |
| | AT&T & Philips Telecom | X | X | TCP 16, TCP 36 |
| Denmark | Technical University of Denmark / Administration / Imperial Software (UK) | X | | |
| Switzerland | Hasler | | X | TTCF Telex switch |

**Table 5.1 Chill implementations and applications.**[64]

As can be seen, most of the table is made up of organisations that had committed themselves to Chill through their active participation in its development. It also tells us that few administrations were directly involved in using Chill, even though some implementations were carried out by administration-affiliated research institutes, such as the Italian SIP-CSELT. A few administrations, such as the Nordic and the Dutch, were directly

[64] Based on information in "Reply to Question 8/XI – Maintenance, training, compliance and environment aspects of CHILL", CCITT Plenary Assembly 7 Yellow Book Vol. 1-3 1980, CCITT, ITUA. Other sources are Svein Hallsteinsen, "Overview of projects and contracts at RUNIT dealing with CHILL and CHIPSY", 20 November 1985; Kristen Rekdal "CHIPSY – Reference List", 6 March 1986, URD Information technology A/S; Kristen Rekdal, "Report from fourth Chill Conference", 8 October 1986, all in "L 0136 Samarbeid", series "Da, 1961 – 1996", NTR. Some details of the outcomes of the joint venture between AT&T and Philips Telecom draw on interviews with Remi Bourgonjon and Kees Smedema. On the Korean use of Chill, see Kwon Yong Rai, "Software technology and industry of Korea: widening horizon and emerging presence" (Orlando, FL, USA). Additional information are drawn from Rekdal, "CHILL - The International Standard Language for Telecommunications Programming".

involved in implementation projects.

The survey reveals broad diffusion of the programming language in geographical terms, as its use spread from the countries first involved in its design to countries with limited involvement in the early period of Chill. Also notable is that the language spread to a number of projects outside the area of public switching for which Chill was originally mandated, like the military communication switch of the Norwegian ITT subsidiary STK. Some of this diffusion was partly due to the reorganisation of firms like Philips and the ITT, involving subsidiaries in various countries and new joint ventures like AT&T Philips Telecommunications and the sale of ITT's telecommunication division to Alcatel. However, its diffusion to countries like India, South Korea, Brazil and China reveals a geographically expanding community: new implementations and applications sprung up in these countries, in particular in the fourth study period within the CCITT, running from 1985 to 1988. The Indian, Brazilian and Korean efforts were also on a large scale, involving the programming of what would eventually become commercially available switching systems.

## Some conclusions

In terms of use by large manufacturers, Chill was a moderate success and a relative failure. Since its inception in the mid-1970s, more than 12,000 programmers worked with the language. Some of the most successful telecom switching systems on the world market until the early 1990s were engineered with Chill, like the System 12 family of switches from the ITT and the EWSD switches from Siemens.[65] If one counted the number of installed lines of public exchanges by the early 1990s, one would find that Chill dominated as the language used by the most installed switches, with the programming language Protel, used by Northern Telecom, and C, used by AT&T, the next most important.[66] Chill was, at that time, the only programming language common to more than one of the major public telecom switching systems.

Chill became the tool of choice in a number of extremely large programming projects, involving large teams, spanning numerous user sites and in organisations that spread their programming across national boundaries, resulting in technically large systems. Furthermore, Chill was

[65] Statistics and numbers are found in Rekdal, "CHILL - The International Standard Language for Telecommunications Programming". The main switching system programmed in Chill was System 12, produced by ITT and Alcatel, and EWSD by Siemens. See Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*.

[66] Rekdal, "CHILL - The International Standard Language for Telecommunications Programming".

adopted by researchers and manufacturers in emerging economies, in particular in China, India and Brazil. Following this, in 1993 Kristen Rekdal concluded: "It is safe to say that CHILL has largely achieved its original objective of becoming a standard language for the programming of public telecom switching systems."[67] However, most of the manufacturers that used Chill were part of the Chill project from its outset, with the exception of the more specialised application of Chill in newly industrialised countries.

The evolution of the telecommunication equipment market fostered, as well as hindered, the widespread use of Chill in the 1980s. For example, the use of Chill in ITT and later on, in Alcatel, was intimately related to the expanding international market, as well as the acquisition of ITT's telecommunication division by Alcatel. An example of the opposite was the case of Philips. In 1983, Philips joined the American AT&T in a joint venture, a move that would eventually halt the use of Chill in that company, although not without a fight: the American AT&T first agreed on developing a system for translating existing software and integrating new Chill-based software into their new switching system. The Siemens use of Chill can be understood in a similar fashion, as it coincided with a sustained effort to export the Siemens systems beyond the company's typical markets, in particular to the US market after the AT&T divestiture in 1984. The adoption of Chill within the ITT was, on the other hand, related to their strategy to commit early to Chill as a way to gain market shares. However, the difference in strategies between ITT subsidiaries highlights the considerable amount of independence lent to technical expertise in technical decisions, and as such at least implicitly underscores the importance of the resonance Chill had among some technical practitioners.

All in all, Chill was really neither a success nor a failure: it was something in between, something that did not really take off, but neither did it fall flat. Most discouraging was the relative failure of the vision of Chill as a tool that would increase the administrations' control over their own equipment and procurements, which would simply disintegrate during the 1980s. Few of the users documented in the survey above had any clear relationship with the administrations, and few administrations would enforce the use of Chill upon their manufacturers, even though many of them acted upon a fear of such enforcement. This limited appeal to the telecommunication administrations is the subject of the next chapter.

[67] Ibid.: 6.

# 6. Advances and rejections: administrations, communities and the struggle for diffusion

Throughout the 1980s, Chill was improved and made more capable. The language specification was refined and new language features were added. In particular, the language was made more advanced and efficient when used in large projects. Despite these advances, the language did not attract any real interest from telecommunication administrations, either as a tool or as a mandatory requirement in their procurement. Consequently, its impact and diffusion was more restricted than the initial success had suggested. This chapter looks at why the national administrations abandoned the language.

Following the rejection by the administrations, the Chill community of language designers became even more dominated by participants from manufacturing firms. Still, the community expanded in real term and went to improve the language considerably. This community was underlined by an infrastructure of conferences, user groups and a circulated bulletin.

This chapter explores the duality of technical improvements and limited diffusion among the administrations, from around 1980 and up until the late 1980s. In particular, I trace the reasons why the administrations rejected the language, and the reasons to continue improving the language within the CCITT and the wider Chill community. Following this, this chapter answers the question on how the pattern of diffusion directed the advances and improvements made to the language.

## Commitments

In December 1980, the telecommunication administrations of Austria, Belgium, Finland, France, the Federal Republic of Germany, Norway, the Netherlands, the United Kingdom, Sweden and Switzerland released a statement on their continued support for Chill in preference to any other high level language for telecommunication applications.[68] The signatories to the agreement stated that "since Recommendation Z.200 was approved unanimously and since the Administrations concerned invariably seek harmonization in agreement with the CCITT, CHILL is the approved

[68] The agreement was revealed in a letter from the president of the responsible CEPT committee, D. Gagliardi, to the European Commission, February 1982. D. Gagliardi to Commissions des Communautés, 3 February 1981, Rome, in Annex 6 to Doc. T(81) 4 Add, "Télecommunications" Réunion extraordinaire de la Commission Innsbruck 11 – 20 mai 1981, Tome II, Documents présentés á la Commission (T (81) 1 á (81) 28)." box "L0022 – Telekomiteen, 1979 – 1984", series "Dbc Utenlandskontoret," Archive "Administrasjonsavdeling", NTA. A report on the meeting is also given in Kristen Rekdal, "Report from CEPT/CCH meeting on CHILL vs Ada, Helsinki 5 December 1980," box "NT-P 1980-1981", KRC.

language within their countries". [69] Evidently, these 10 European administrations wanted to put their weight behind Chill, by stating it was *the* approved programming language in their territories. The agreement was released after a meeting in a special working group of the CEPT, the European Conference of Postal and Telecommunications Administrations, which until the early 1990s was engaged in cooperation on commercial, operational, regulatory and technical standardisation issues, in many ways a European ITU, even more "administration dominated" than the CCITT.[70] Support from the CEPT reaffirmed the initial impression of Chill as a tool favoured by the administrations, and also hinted towards the idea of programming as an activity carried out by the administrations. The statement is also in line with those made at the inception of the Chill project, when it was proposed that the administrations should take larger responsibility for the software of their switching equipment. By the late 1970s and early 1980s, similar tendencies were reported as emerging in Australia, New Zealand and in smaller countries like Singapore.[71] In a paper presented at the 1981 SETSS conference, the importance of gaining control over the software was emphasised by Finnish administration representatives: "Trying to gain independence from the manufacturers, the administrations are very interested in the production and maintenance of the software they need."[72]

The CEPT statement was, however, not only an expression of the proactive vision of the administrations. It was also a response to increasing pressure to commit to the programming language Ada. During the first years of the 1980s, the momentum behind Ada was increasing, as it gathered support from the European Economic Community (EEC) and a number of industry actors, and it was believed to be paramount to the future of Chill to fend off the competition. It was clear that in technical terms, the two

---

[69] "Programming languages for telecommunication applications", COM-XI Temporary Document No. 36-E, Geneva, 6-16 April 1981. Published in *Chill Bulletin* 1, no. 1 (1981), 40.

[70] On CEPT, see Gerhard Fuchs, "Policy-making in a system of multi-level governance-the Commission of the European Community and the restructuring of the telecommunications sector", *Journal of European Public Policy* 1, no. 2 (1994).

[71] Kristen Rekdal, "Reiserapport fra CCITT Implementors Forum, CCITT WP XI/3, TELECOMM Australia, University of Canterbury, Bell Labs, 20/9 – 16/10 1979," box "Implementors Forum 9. møte Melbourne, Sept. 1979, Serie O", KRC. On the Singaporean case, see Kristen Rekdal, "Travel report from 1. Telecoms, Singapore, 2. CCITT WP VII/3, Melbourne, 3. Nord Computers, Melbourne, 4. Intel Corp, Santa Clara", Runit notat, 13 April 1982, box "NT-P 1982", KRC.

[72] J. Hirvensalo, A. Myllkangas, and K. Rahko, "Quality standardization of telecommunciation swicthing system software", in *Software Engineering for Telecommunication Switching Systems* (University of Warwick, Coventry, United Kingdom: Institution of Electrical Engineers, 1981), 16.

languages were quite similar and that the battle for credentials was just as much a battle for political and commercial support.[73]

One month before the CEPT meeting, the plenary assembly of the CCITT had finally approved the Chill recommendation. However, neither here, nor at the CEPT meeting in December 1980 did this happen without a quarrel. While the recommendation was approved without any problem, the new mandate for the working group responsible for Chill caused trouble. The French administration had filed a proposal that would instruct the new working group to reconsider the "technical and economic criteria for determining the preferential applications of Chill" given that "other high level languages for similar purposes already exist and action has been taken to extend the field of use of one of them, namely Ada, to telecommunications".[74] In non-diplomatic terms, this proposal implied that the CCITT should consider in which cases, if any, Chill would be given preferential treatment. After numerous rounds of corridor diplomacy, the choice of words was altered slightly, as Ada was only mentioned as an example and references to technical and economic criteria were deleted.[75] While Ada had crept into the CCITT papers, it was not longer *the* reason for further study into which areas Chill would be a preferred technology.

Yet, only a month after the skirmish at the CCITT plenary assembly, the same strategic manoeuvre was repeated at the CEPT meeting mentioned above. Once again, the French administration was eager to promote Ada as a viable alternative to Chill. Their preferred route of action was through the constitution of a working group that should report on the possibility of "harmonisation" of the two languages.[76] Three official documents were presented at the meeting, one by the French administration, one by the Norwegian and one by the Dutch. The two latter documents presented pro-

[73] Some comparisons are found in Smedema, Medema, and Boasson, *The programming languages : Pascal, Modula, CHILL, and Ada*; Erik Meiling and Steen U. Palm, "A Comparative Study of CHILL and Ada on the Basis of Denotational Descriptions", in *Second CHILL Conference* (Lisle, Illinois: Bell Laboratories, 1983).

[74] "Draft addendum to new question 8/XI", 18 November 1980, Temporary Document No. 18-E/COM B, box "NT-Programspråk 1980-81", KRC.

[75] Nic Knutzon, "VII CCITT plenarforsamling, Behandling av CHILL spørsmålet", 24 November 1980, box "NT-Programspråk 1980-81", KRC.

[76] Administration francaise, "Harmonisation de langages de programmation de haut niveau", box "NT-Programspråk 1980-81", KRC.

Chill views.[77] The outcome was the agreement mentioned above, as the delegates to the CEPT meeting concluded that the organisation was not "interested in, or in a position to, take part in the harmonisation of other languages".[78] The report by Kristen Rekdal, who participated at the CEPT meeting, stated that the French administration felt very strong pro-Ada pressure from French industry, which was perhaps no surprise, given that the language was developed at CII Honeywell Bull. The atmosphere was also influenced by the EEC, as the Commission encouraged the setting up of a specific interest group, called Ada-Europe, which brought together technical expertise at a European level and exerted certain influence over the development of the language. The hope of the European Commission was that these activities and support programmes would:

> encourage the European industry would commit itself more firmly to the development of completely new software technology stemming from Ada, on the same line as the energetic efforts being made by the American industry, research institutions and universities.[79]

As discussed in chapter four, the funding directed towards Ada-oriented projects was substantial. One of the projects that benefited was carried out by GEC telecommunications in the UK and the Dansk Datamatics Center, which had been started by Dines Bjørner. This was to study the support for Chill regarding a future "Ada programming support environment."[80] Despite this small conciliatory gesture, the telecommunication administrations and the industrial partners involved in the Chill project looked on the EEC initiatives with scepticism. In a reply to a letter sent by the Dane Jens Rasmussen, of the Nordic Chill project, one of the Commission's bureaucrats replied that they were "aware of Chill", but their support for Ada was based on its "standards aspects and potential effects on market and industrial structure".[81] Accordingly, Chill was not regarded as a language that could provide the same effects. The letter continued as follows:

---

[77] Administration francaise, "Harmonisation de langages de programmation de haut niveau", box "NT-Programspråk 1980-81", KRC. Netherlands PTT, "Harmonization in the field of SPC programming: CHILL, Ada, or both?", Doc T/CCH (80) 17, box "NT-Programspråk 1980-81", KRC. Norway, "CHILL, ADA and ESL", Doc T/CCH(80)18, box "NT-Programspråk 1980-81", KRC.

[78] Kristen Rekdal, "Report from CEPT/CCH meeting on CHILL vs. ADA", 5 December 1980, box "NT-Programspråk 1980-81", KRC.

[79] "Community Data-processing Policy", Communication from the Commission to the Council, Brussels, 22 July 1982, 23.

[80] Ibid., 40.

[81] H. Hünke to Jens R. Rasmussen, Brussels 30 August 1982, box "NT-P 1982", KRC.

> Our requirements for standards come from one of the major goals of the European Economic Community for open markets and free (i.e. unhindered) exchange of goods and services. Only standards assuring a very high degree of portability can be expected to contribute towards this goal.[82]

Obviously, open markets and free exchange of goods and services would not be considered as a likely outcome if Chill were made mandatory as a standardised programming language for telecommunications, according to the EEC insiders. In brief, it was considered a property of the "ancien regime" of telecommunications rather than a technology that could break it. The pressure from the EEC would, at first, force the CEPT to encourage the industry to use Chill, and did not alter its preference for Ada, quite contrary to the intentions of the EEC.

The attitude of the administrations was a lot more ambiguous than the CEPT agreement made it seem. The stipulation of Chill as a prerequisite for tenders was generally not followed up. Neither was the idea of software development and maintenance done within the administrations. In the next section, I will closely follow this ambiguity in the case of the Norwegian administration, and contrast this with the Swedish administration, which abandoned all links to Chill in the mid-1980s and embraced Ada wholeheartedly. Both administrations had been particularly important in the first years of the Chill project, and therefore make up an interesting pair of cases when considering the ambiguity and negativity that the telecommunication administrations felt towards Chill in the first half of the 1980s.

## Ambiguity and negativity

The case of the Norwegian telecommunication administration (NTA) illustrates both the hopes the onset of ambiguity. In late 1979 the policy of the administration was that of enforcing Chill on its prospective suppliers. A policy note circulated within the administration in October 1979, stated: "We find it important to make clear that [the NTA] wants to use Chill in future switches and other processing equipment for the telecommunication network and that we will make this a requirement in future specifications of such equipment."[83] The note added a hope that the Norwegian industry would follow and use Chill as its programming language. This started a long discussion about the role of Chill in a future tender for the digital backbone

[82] Ibid.

[83] TAS/79/Kha, "Vedr. CCITT høynivå programmeringsspråk, CHILL" 3 October 1979, Notat fra Teledirektoratet, box "NTT 72-2/NT-P 1979 – 1980", KRC. My translation.

of the Norwegian telecommunications network.[84] In this discussion in the early 1980s, the ideal of a Chill-programmed system was put to a real test: L. M Ericsson had made it clear they did not intend to offer a system programmed in Chill. They would stick with the proprietary language Plex for the moment but might opt for Ada at a later time. L. M. Ericsson was also the favourite supplier of many in the technical division of the Norwegian telecommunication administration, not least after the problems the administration had experienced with their last large-scale procurement of switches from the ITT. The technical director of the NTA, Ole Petter Håkonsen, wrote the following:

> Even if we introduce a requirement in terms of use of Chill in our specifications on digital switches now, it seems obvious that we are not in a position where we can exclude well known systems developed in another language. However, such a requirement should indicate to our supplier that the next generation of their systems should preferably be made with Chill.[85]

In the same note, Håkonsen also admitted that if they were to choose a supplier that did not use this international standard, it would be a failure:

> As particularly active in this field, it would hurt our credibility if we do not use this recommendation. A lot of manufacturers of telecommunication equipment have already implemented it and it is only fair that they get the support of the administrations.[86]

By 1982, the NTA was ready to sign a contract to purchase a number of new digital switches, after much political and economic wrangling. According to historian Lars Thue, the decision marked the beginning of a new type of industrial telecommunication policy in Norway, as the tender was open to international bidders and not just the two national, although internationally owned, producers Elektrisk Bureau (EB) and STK.[87] This decision involved technical conflicts in the administration as well as a large political shift, which involved a newly formed conservative government in 1981. The choice would eventually be System 12 from the ITT. However, the contract

---

[84] Detailed chronicles of the various projects leading up to the digitalisation of the Norwegian telecommunication network are found in Christensen, "Switching Relations: The rise and fall of the Norwegian telecom industry"; Lars Thue, *Nye forbindelser: 1970-2005* (Oslo: Gyldendal, 2006).

[85] Ole Petter Håkonsen, ”(Hvorfor Chill?)”, box ”L0048, Indig-prosjektet, 1981 – 1983”, series ”Da Teletjenesteavdeling (T), 1968 – 1992”, Archive ”Teknisk avdeling / Teletjenesteavdeling (T),” NTA. My translation.

[86] Ibid. My translation.

[87] Thue, *Nye forbindelser: 1970-2005*, 147.

was *not* won through a tender from which L. M. Ericsson had been excluded due to their choice of programming language. EB was invited to submit a tender based on the Ericsson-produced switch AXE, programmed in their own proprietary language Plex. The choice of the ITT had, in the end, very little to do with choice of programming language. Still, the technical director Håkonsen managed not to lose face since the administration ended up with a switch that was programmed in Chill.

Coincidentally, the Norwegian administration would continue to support the Nordic Chill project and eventually Urd, the company spun off from their partner, Runit, to which I will return in the next chapter. At face value, the Norwegian administration was backing the international standard Chill, through procurement as well as research and development. However, the realities were far more ambiguous. Throughout the 1980s, the "Chill question" was raised within the Norwegian administrations numerous times. It was proposed in 1986 that the Norwegian telecommunication administration should "concentrate" on Chill and the specification language SDL, a proposition coming from the research establishment of the administration. The SDL policy was actually adopted, but concerning Chill, the technical division of the administration was lukewarm.[88] Again, the possibility of excluding technology from L. M. Ericsson made such a policy unpopular. In 1989 a multidivisional working group within the administration cooperated with industry players, and developed a policy of focussing on Chill within the areas of broadband communication and management networks. The results were meagre – as it continued to be a policy of little more than wishful thinking and had few implications. The Norwegian Telecommunication Administration was ambiguous about Chill throughout the 1980s, even though the impression was that the administration held the language in high esteem.[89]

## Betrayed from within

A comparable story is that of Sweden, although in that case the ambiguity was replaced by outright negativity. Sweden had been one of Norway's partners in the Nordic Chill projects from the mid-1970s and in many ways it was also the main initiator of the CCITT's surge in interest in language design in the late 1960s. The Swedish administration had also sponsored the Nordic compiler projects – and had sponsored the Nordic representative in the CCITT, Kristen Rekdal, throughout his stint there. Still the Swedish telecommunication administration and L. M. Ericsson abandoned Chill almost altogether right after the language was officially endorsed by the

[88] F89/u/949, "Foreløpig anbefaling om bruk av Chill in Televerket", 23 August 1989, box "L 0136 Samarbeid", series "Da, 1961 – 1996", NTR.
[89] Ibid.

CCITT in 1980. In the following, I will analyse this together, which makes sense in the case of Sweden, where the cooperation between the national champion Ericsson and the telecommunication administration was so extensive that they had a joint research establishment, Ellemtel.

L. M. Ericsson had made a considerable contribution to Chill by adding the signals concept to its repertoire of concurrency-related concepts. This concept was, again, derived from their own proprietary programming language, Plex. However, the designer of Plex, Göran Hemdal, was never interested in Chill. In retrospect, he would denounce the project on the basis that it had turned into something completely dominated by computer scientists and programming language theoreticians. To Hemdal, Chill lacked features that would make it work in a telephony setting, and this was due to the composition of the working groups.[90] To Hemdal, the success of Plex was related to the fact that he, as the language designer, really did not know what programming was.[91]

By 1979, Hemdal had moved on to ITT and it was time for change when the APN 167 processor was introduced to the Axe system.[92] However, the result was EriPascal, a Pascal-inspired language, and not a fully fledged Chill adoption. According to Bjarne Däcker, who designed EriPascal, this happened in a rather arbitrary manner: EriPascal was really similar to a subset of Chill, but with a Pascal-like syntax. It included the signals concept from Chill, but not all the other concurrency-related features of the language. The Pascal syntax was chosen because the group responsible for programming technology at L. M. Ericsson was in a hurry and adopted a compiler developed for a Pascal dialect called San Diego Pascal. Instead of making the compiler accept Chill code, they created a programming language that looked like Pascal. The idea was to make the compiler work on Pascal-like code, but also to make it accept "something that looked like Chill".[93] The plan was also to move gradually to a more proper Chill subset and a precompiler for Chill, known as EriChill, "but there were no user requests for it", according to Däcker.[94] Däcker would go on to be a founder of the Ada user group in Sweden in 1983, but would later work on a new specialised programming language for telecommunication inside Ericsson, a

[90] Vedin, *Teknisk revolt: Det svenska AXE-systemets brokiga framgångshistoria* 159.
[91] Ibid.
[92] Peter Magneli, "Communications Computer APN 167 with ERIPASCAL", *Ericsson Review* 63, no. 4 (1986).
[93] Lundin, "Tidlig programmering : Transkript av ett vittnesseminarium vid Tekniska museet i Stockholm den 16 mars 2006", 38.
[94] Bjarne Däcker, "Concurrent Functional Programming for Telecommunications: A Case Study of Technology Introduction" (Royal Institute of Technology, 2000), 11.

language called Erlang.[95] By 1981, the interest in Chill at L. M. Ericsson was close to zero, as was explained in a meeting between representatives of the company and participants in the Nordic Chill, coming from the administrations. L. M. Ericsson made it clear the company would not change its programming language to either Chill or Ada, unless it was forced to do so.[96]

Ada caused a stir not only with Bjarne Däcker in Ericsson, but also in Ericsson's close ally, the Swedish telecommunication administration. One paradoxical figure who was instrumental in the change of policy towards Chill was the Swedish electrical engineer Kurt Katzeff, who was one of the main designers behind Ericsson's Axe switches in the 1960s.[97] Katzeff moved on to the ITT's European headquarters in Belgium in the early 1970s, and was their chief technical officer until he returned to be deputy head of the technical division of the Swedish telecommunication administration from 1980. As Katzeff returned to Sweden, the Swedish administration opted against the use of Chill on his recommendation. Obviously, his stint at the ITT had not made him warm to the idea of Chill as a standard, even though the company was one of Chill's main supporters and came to be one of its main users. Instead, Katzeff argued that the Swedish administration could not "introduce Chill".[98] He felt it was unimportant which programming language was used in future switching systems, as long as the system was well supported and documented by its manufacturer. He claimed it was not the programming language that should be the decisive factor. "Given the alternatives of one system coded in Chill with no support system and one assembly coded system with a powerful support system available, the choice is not difficult," Katzeff wrote, implying that the assembly-coded system would win hands down every time.[99] As such, it was all up to the manufacturing firm. Furthermore, Katzeff was altogether uncertain about the necessity of such programming languages as Ada and Chill:

---

[95] The Swedish Ada-user group was started in 1983. See Lundin, "Tidlig programmering : Transkript av ett vittnesseminarium vid Tekniska museet i Stockholm den 16 mars 2006", 34.

[96] Knut Bryn, "Diskusjon mellom L. M. Ericsson og NT (Samordningskomitéen for Teletekniske spørsmål unntatt readiotekniske) vedrørende CHILL," 5 October 1981, box "NT-P møte 8 og 9", KRC.

[97] See Lundin, "Tidlig programmering : Transkript av ett vittnesseminarium vid Tekniska museet i Stockholm den 16 mars 2006", 31.

[98] Kurt Katzeff, "The use of high level programming languages in the field of telecommunications", *Tele* 1981(2), 7-11.

[99] Kurt Katzeff and Anders Rickström, "Software standards in the field of telecommunications", in *Fifth International Conference on Software Engineering for Telecommunication Switching Systsems* (Lund, Sweden: Institution of Electrical Engineers, 1983).

> […] the apparent conclusions of most comparisons to date are, that the computer world does not need two such similar languages as Chill and Ada, and that they may not be ready for practical use until it is too late, as advances in specification and description language, support tools and systems architecture, may well make both languages redundant.[100]

Despite this, the Swedish telecommunication administration would soon put all its weight behind Ada. In 1983, the Swedish administration and Katzeff spun out a company named Telelogic, a research and development company focusing on software and development tools.[101] At the same time, a policy of adopting Ada, with an exception for the Axe sphere of L. M. Ericsson switches, in all future products, was put in place.[102] Two years later, Telelogic acquired parts of the American company Telesoft – an independent developer of Ada tools - to strengthen their position as an important player in the market of firms making tools for Ada. In 1984, Ericsson sat up the Computer Science laboratory – which would be engaged in the design of an altogether new programming language, Erlang, which was planned to be a successor of Plex for the Axe sphere.[103] Both events were strong indications of the Swedish reluctance towards Chill, which manifested itself in the non-use of Chill in every part of the Swedish telecommunication system. Taken together, the Swedish case is one of abandonment. By 1984, the Nordic cooperation on a common Chill policy ended, and according to the Swedish administration representatives, the compiler developed by Runit had "so far found no use in the Swedish administration".[104]

Still, two years later, in 1986, Ericsson would reconsider the use of Chill in Axe.[105] This time around, the idea would be dismissed not because of enthusiasm for Ada, but because of Ivar Jacobson, inspired by object orientation.[106] Jacobson had participated in the Chill work on behalf of

---

[100] Kurt Katzeff, "The use of high level programming languages in the field of telecommunications", *Tele* 1981(2), 7-11.

[101] Lundin, "Tidlig programmering : Transkript av ett vittnesseminarium vid Tekniska museet i Stockholm den 16 mars 2006", 31.

[102] *Tele*, 1988 (2), 22 – 25.

[103] On Erlang, see Däcker, "Concurrent Functional Programming for Telecommunications: A Case Study of Technology Introduction".

[104] A. Rockström, "Future work-items for NT-P: Swedish point of view", 2 February 1984, box "L 0135, Samarbeid", series "Da, 1961 – 1996", NTR.

[105] Ivar Jacobson, interview with the author, 22 February 2011.

[106] I will return to the subject of object orientation later in the chapter. Here, it suffices to say that this way of programming differed substantially to what was common practice at the time.

Ericsson in the late 1970s, most significantly contributing the concurrency-related signals concept to the language.

By 1983, Jacobson spent a year at MIT on sabbatical leave, preparing for his doctoral thesis, by reading up on the latest advances in computer science, in particular object orientation. This would significantly influence Jacobson's later work, also his doctoral thesis, which was defended at the Swedish Royal Institute of Technology in 1985.[107] By 1986 he had met Tom Love, who had just developed the language Objective-C (together with Brad Cox), an object-oriented extension of C.[108] Inspired by Love and a lot of reading about Smalltalk and object-oriented programming during his year at MIT, Jacobson started working on a proposal to extend the existing Ericsson programming language Plex into Objective-Plex. On his return to Sweden, Jacobson was consulted on the issue of using Chill in the AXE line of switches. Ericsson had, as noted above, got their way in terms of communication mechanisms in Chill and in some parts of the firm they were now apparently eager to use the CCITT standard. Jacobson remarked that it would be much cheaper, faster and more effective to go for Objective-Plex. Following this, Ericsson put the lid on any future Chill plans, and worked on the Objective-Plex path. Objective-Plex was a simple extension of Plex, and could have yielded rapid results. However, after the beginning of the work to objectify Plex, another path was selected towards C++. All in all, Jacobson's arguments to objectify Plex were what really ended all possibilities of the use of Chill at L. M. Ericsson.

A similar tendency to the Swedish experience was under way in Japan by the mid-1980s. The role of the telecommunication administration of Japan, the NTT, had been highly visible in the development of Chill from the beginning. As noted in the previous chapter, Chill was used by the NTT and some of its industrial partners in the development of switching equipment. From 1979, the NTT had used Chill when developing the D10 switch.[109] The international telephone carrier of Japan, the Kokusai Denshin Denwa (KDD), also applied Chill in switching projects in cooperation with NEC.[110]

However, there is also evidence that the Japanese telephone companies faltered in their support for Chill. While the D10 switch was

---

[107] Ivar Jacobson, "Concepts for Modeling Large Real Time Systems" (The Royal Institute of Technology, 1985).

[108] At the time Objective C was not really a separate language, but a language extension with a pre-compiler for C.

[109] Maruyama, Sato, and Konishi, "NTT CHILL implementation aspects and its application experience". See also Larsson, "Future Telecommunications in Japan - Policy and Technology".

[110] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 426.

developed in Chill, there was still widespread uncertainty within the NTT on what to do with Chill after it was ratified as a standard in 1980. Future support for the language could easily ceased right after that, according to Norio Sato, who became the NTT's delegate to the CCITT from 1981. Real uncertainty about the future of Chill existed. However, it was decided to develop its use further towards more modern environments, microprocessors and new switching equipment, which continued throughout much of the 1980s.[111]

However, from the mid-1980s, the NTT's interest in Chill waned. This was related to two major events: the privatisation of the NTT in 1985 and the initiation of the large research project Tron in 1984.[112] While the former only influenced the use of Chill in the NTT indirectly, the latter had a more direct and severe influence. I will deal briefly with each.

Traditionally, the government-owned NTT had designed, developed and operated the Japanese domestic telecommunication network. At the same time, the NTT played an important role in researching and developing new telecommunication technologies. To some extent, this role was performed in cooperation with the group of competing suppliers like NEC, Fujitsu, Hitachi, and OKI Electric, which over time had been engaged in some sort of coordinated competition. To some extent, the 1984 part-privatisation and the introduction of competition in both long-distance and local telecommunication services changed the "NTT way", both in terms of operations and in terms of research and development. In the long term this meant that the supplier companies would have to rely on their own research and development rather than that done by the NTT. The R&D expenditure of the NTT actually increased in the aftermath of the privatisation and the opening up to competition. However, the NTT's R&D priorities changed from equipment to network planning, design, operations and new services, which at least indirectly influenced the switching projects to which Chill had been applied, which were tied to equipment development.[113]

---

[111] On this, I rely on Norio Sato, emails to author, March 2011.

[112] On the privatisation of the NTT, I rely on Fransman, *Japan's computer and communications industry : the evolution of industrial giants and global competitiveness*; ———, *The market and beyond : information technology in Japan*. To the best of my knowledge, no real comprehensive discussion of the TRON project exists. For fragmentary descriptions and discussions, see Marie Anchordoguy, "Japan's software industry: a failure of institutions?", *Research Policy* 29, no. 3 (2000); Takuma Takahashi and Fujio Namiki, "Three attempts at "de-Wintelization": Japan's TRON project, the US government's suits against Wintel, and the entry of Java and Linux", *Research Policy* 32, no. 9 (2003).

[113] Fransman, *Japan's computer and communications industry : the evolution of industrial giants and global competitiveness*.

The fate of Chill in the NTT was apparently more directly influenced by the initiation of the so-called Tron project, a large and ambitious research project introduced with great fanfare in 1984.[114] The project's general goal was to replace American software and hardware technologies with home-grown ones, involving everything from an operating system to integrated circuits and processors. The project was initially a collaboration between a number of private companies and was led by professor Ken Sakamura of the University of Tokyo. However, it also influenced a number of choices made at the NTT. Among them was the decision to develop an operating system for switching systems based on technologies and specifications defined by the Tron project, a project that was in turn named CTRON.[115] This meant that future switches would have to rely on the programming language C and not Chill. This was quite a paradox since C was an American programming language directly related to the Unix operating system, controlled by the AT&T. The Unix system was one of the direct causes behind the anti-American objectives of the whole TRON project.[116] Nevertheless, the overarching concepts and rationales behind the TRON project led meant less work was put into Chill diminished and future programming on new switching projects would be done in C. By December 1993, Chill developments within NTT were almost at an end.[117]

Summing up, it seems fair to describe the position of the administrations as more passive in terms of enforcement than what was hoped in the 1970s, and feared by the manufacturers in the 1980s. As such, Chill failed as an administration-driven wedge between operators and manufacturers. However, this wedge was thoroughly enforced by political bargaining, as efforts to liberalise and reorganise the sector swept across Europe, America and Japan during the 1980s and 1990s.

## Modular improvements

Despite the gradual defection of the administrations, Chill was continually worked on and improved in the first part of the 1980s, resulting in a significantly improved version published as a new recommendation in

[114] A hyperbolic description of the project written by the project leader is Ken Sakamura, "The TRON Project", *Information and Software Technology* 38, no. 3 (1996).
[115] Information about this is of limited availability. A virtual computer museum set up by the Information Processing Society of Japan (IPSJ) holds some information about the NTT's involvement in the TRON project. "NTT CTRON", IPSJ Computer Museum, http://museum.ipsj.or.jp/en/computer/os/ntt/0078.html. (retrieved 8 April 2011).
[116] Norio Sato, e-mails to the author, 30 September 2008.
[117] Norio Sato, e-mails to author, March 2011.

1984.[118] In practical terms, improvements made to Chill were discussed within a sub-working party in the CCITT, which came up with a revised language definition, which in turn was ratified by the CCITT plenary in 1984. This resulted in the publication of the revised edition of the language specification, the Z.200 document. The sub-working party responsible for Chill improvements, the "Sub-Working Party XI/3-2" in the CCITT hierarchy, had several agendas and goals. Improvements made the language more capable in large-scale projects. Furthermore, standardised facilities for input and output were added, meaning a set of language features that would make it easier to provide data transfer between Chill programs and its environment. While all this might sound trivial – both were big deals. Both areas were considered weak spots by the large firms that already had used the language, as evident in the ITT story discussed in the previous chapter.

These improvements pay testimony to the considerable influence the firms using the language in the development of real switches held at the time. The addition of facilities for input and output meant that set features would make it easier to provide data transfer between Chill programs and its environment. This included features for the manipulation of files and records within files, which in general would make it easier to use Chill in real-world projects, or at least would make it unnecessary to create such facilities time and time again.[119] The second area of improvement concerned facilities for separate compilation. In the lingo of the language designers these was called facilities for "piecewise programming". This involved the possibility of separate compilation of program modules.[120] Piecewise programming allowed independent development and compilation of pieces of a program, something that was important for the use of the programming language in projects involving a large number of programmers, who could now safely code away on their respective bits and pieces. Thus piecewise programming was a technological improvement that allowed the possibility to the division of labour in large programming projects.

The details about the processes leading up to these improvements are sketchy. Both areas of improvements added functionality and concepts that had long been discussed within the Team of Specialists and in the Implementors' Forum. In the two previous periods (from 1974 and up until 1980), the separate compilation element was thought of as a technological hurdle of such proportions that it could not be dealt with adequately in the available time. It was a deliberate leftover and something that the language

---

[118] *CCITT High Level Language (CHILL)*, CCITT Recommendation Z.200 (1984).
[119] "Draft Proposal for Basic Input/Output Facilities in CHILL," *Chill Bulletin* 3, no. 2 (1983).
[120] "Report on CCITT SWP XI/3-2 Meeting in Geneva December 1981," *Chill Bulletin* 2, no. 1 (1982).

designers thought could be treated on a level different from the language definition. By 1981, the language designers would think differently in the last respect, namely that the issue of separate compilation was something that would have to be implemented in the language itself. The problem of using Chill when doing "programming-in-the-large" was very much the constituting agenda when discussing separate compilation facilities.[121]

From 1981, compilation issues were given considerable attention in the CCITT meetings. Initial contributions were made by the French and the Italian administrations, Runit of Norway and the NTT in Japan. Other propositions came from AT&T (this was before the joint venture with Philips), British Telecom and Philips, which quickly formed an alliance to create a common platform.[122] The alliance formed around three different proposals, which all tried to rectify the shortcomings in the programming language in terms of dealing with the development of large programs: AT&T proposed some extensions for controlling the visibility of names and enhancing the facilities for libraries in Chill. British Telecom was about to embark on a project involving multisite development and the design of a programming support environment for both Chill and Ada and proposed extending Chill with Ada-like facilities for separate compilation. Philips proposed extending Chill with a set of facilities that provided for the specification, decomposition and manipulation of large pieces of code. Before a meeting in Geneva in December 1982 the three organisations agreed on a common proposal that incorporated the different propositions, which eventually received a wide acceptance from the participants in the working group.[123] Here, the experience and learning of different organisations fed back into the core Chill development group and clearly influenced the final solutions. This was a reiteration of a common strategy in the Team of Specialists: alignment through alliances and common proposals. However, this time around, the proposals stemmed from organisations that either had gained experience from using the language in systems programming, or at least in implementing the language and not by aligning ideals and virtues.

It is worth highlighting that at this time, some administrations were still taking part in the CCITT work, as the presence of British Telecom, the French and Italian administration-run research establishments and the

---

[121] "Report on CCITT SWP XI/3-2 Meeting in Geneva December 1981," *Chill Bulletin* 2, no. 1 (1982).
[122] R. Bishop et al., "Separate Compilation and the Development in Large Programs in CHILL", in *Fifth International Conference on Software Engineering for Telecommunication Switching Systems* (Lund, Sweden: Institution of Electrical Engineers, 1983).
[123] Ibid.

administration-allied Runit of Norway. Furthermore, participants from the Danish administration-run research establishment was an important part in the work on the piecewise compilation work. Still, the impetus behind these changes was very much rooted in the needs of large programming efforts by the manufacturing firms.

The propositions that in the end led to these improvements were very much also a part of the larger Chill community. Some of the papers at the Chill Conference in 1983 and at the larger International Conference on Software Engineering for Telecommunication Switching Systems the same year dealt with these issues.[124]

Beyond the first half of the 1980s, Chill was revised continuously up until the late 1990s.[125] Again, a dominance of industry participants was evident. One of the most decisive changes was the gradual evolution of so-called *object-oriented* language concepts in the standard, a major influence on almost all programming language designs from about the early 1980s. What exactly entails the term *object-orientation* is disputed.[126] That it signified a considerable shift in programming language design is, however, quite obvious: the historian of programming languages, Mark Priestly, has argued that a definite shift in programming language design happened with the advent of *object-oriented* programming languages in the early 1980s.[127] This was bound up in an underlying change in what was considered virtuous programming, a change in the dominant programming virtue. Much of the mathematical computer science had been spun around the Algol programming language and originated in the tradition of scientific programming carried out in the 1950s. The background for the constitutive *object-oriented* programming language, Smalltalk, on the other hand, was a completely different conception of what programming should be. As Priestley has argued for Smalltalk: "Programming was conceived not as the production of code following an engineering-like process, but as an ongoing interaction with a complex and reactive system."[128] This would, eventually, radically influence the practices of software engineering, and by the 1990s: "Object-oriented technology has become a dominant – if not the dominant –

[124] Reithmaier, "Compilation Control in a Large CHILL Application"; A. Rudmik and B. G. Moore, "The Seperate Compilation of Very Large CHILL Programs", in *Second CHILL Conference* (Lisle, Illinois: Bell Laboratories, 1983); Jurgen Winkler, "A New Methodology for I/O and its Application in CHILL", in *Second CHILL Conference* (Lisle, Illinoise: Bell Laboratories, 1983).
[125] Described in Winkler, "CHILL 2000".
[126] Deborah J. Armstrong, "The quarks of object-oriented development", *Commun. ACM* 49, no. 2 (2006).
[127] Priestley, "Logic and the development of programming languages, 1930 - 1975".
[128] Ibid., 214.

software technology."[129] To some of the practitioners in the field, it had proportions similar to that of Kuhn's paradigm shifts. To Brian Cox, who developed Objective C together with Tom Love at the ITT, it was a "paradigm shift – a software industrial revolution".[130] The euphoria of the rhetoric was related to the perceived otherness of object-oriented programming when compared with regular programming – and the otherness in terms of programming language design: basically, object orientation meant that data structures and the algorithms used to manipulate the data could be presented to programmers as a single entity. However, this little distinction had larger implications. In traditional programming languages, the structure of the language closely modelled the computer, as it split features for expressing algorithms and features for describing data structures, mirroring the split between the data store and the control and arithmetic units in the dominant computer architecture.[131] In object-oriented programming, the form of the programming languages differed profoundly from those developed to mirror the scientific computing models of the Algol kind.

The object-oriented approach made inroads in the Chill community by 1984, four years after Smalltalk had reached its definitive form.[132] Subsequently, both Siemens and Alcatel (after its acquisition of the telecommunication division of the ITT) developed their own variants of Chill that included object orientation features.[133] By the mid-1990s, this interest was reworked into the official language definition of Chill.[134] All in all, the addition of object orientation to the repertoire of Chill pays testimony to two elements: the changing virtues of computer science and the growing interest in object orientation in industry. The move towards object orientation could be understood as a move away from the dominant doctrines of the 1960s and 1970s.

By the time Chill officially got object-oriented concepts in its official recommendation, the language was, however, far less viable than 10 years before. By this time, most telecommunication administrations were

---

[129] Hugh Robinson and Helen Sharp, "The emergence of object-oriented technology: the role of community", *Behaviour & Information Technology* 28, no. 3 (2009).
[130] Brian Cox, "There is a silver bullet", *Byte* 1990.
[131] Priestley, "Logic and the development of programming languages, 1930 - 1975", 219.
[132] J. F. H. Winkler, "The Realization of Data Abstractions in CHILL", in *Third CHILL Conference* (Cambridge University: ITT Europe, 1984).
[133] A. Scrotesse, "OO_CHILL: Integrating the object paradigm into CHILL", in *Fifth CHILL Conference* (Rio de Janeiro1990); Georg Diebl, Georg Schulz, and Jürgen F. H. Winkler, "Object-CHILL: The Road to Object Oriented Programming with CHILL", in *Fifth CHILL Conference* (Rio de Janeiro1990).
[134] *CHILL - The ITU-T Programming Language*, ITU-T Recommendation Z.200 (1999).

transformed into network operators competing in a liberalised market, with considerably different strategic priorities than integrating more technical expertise into their organisations.

The first wave of improvements made to Chill, made in the first half of the 1980s, was made in a period where some administrations still followed the CEPT agreement on continued support for Chill in preference to any other high level language for telecommunication systems.[135] In the case of piecewise compilation, the agreement that was necessary involved work by participants from administration-related research establishments, in particular the Runit and the Danish telecommunication laboratory, and some direct involvement of the British telecommunication administration. Nevertheless, the technical diplomacy behind the improvements made in the first half of the 1980s was intimately related to implementations and knowledge gained through use, which was dominated by implementation by the large manufacturers, in particular the ITT, Siemens, Philips (later on together with AT&T). This dominance would be reflected in the Chill community, which is the subject of the next section.

## Coordinated emergence

In the Team of Specialists and the Implementors' Forum, identities and virtues clashed. The period beyond 1980 was one of more unity, with the emergence of what can be described as the Chill community. The Chill community can be envisioned as being made up of two parts. Firstly, the actors associated with the formal CCITT working group were at its core. Secondly, actors that were involved in the use and development of Chill *outside* the ITU tower of Geneva were part of the wider Chill community. Both the core and the periphery of the Chill community consisted of actors working for telecommunication administrations, manufacturing firms or research institutions, some of them even working in competing organisations at the local level. The international and transnational meeting points of the CCITT, conferences and transnational user groups made up a level for interaction where the local user sites were less important and explicit than they otherwise would have been, although the local user sites were the starting point for the interactions and the decisions made within the Chill community.

The Chill community of users, prospective users, language designers and researchers was partly constructed by the CCITT. Coordinated efforts

[135] D. Gagliardi to Commissions des Communautés, 3 February 1981, Rome, in Annex 6 to Doc. T(81) 4 Add, "Télecommunications" Réunion extraordinaire de la Commission Innsbruck 11 – 20 mai 1981, Tome II, Documents présentés á la Commission (T (81) 1 á (81) 28)", box "L0022 – Telekomiteen, 1979 – 1984", series "Dbc Utenlandskontoret", Administrasjonsavdeling, NTA.

structured the way the community functioned, as the CCITT set up a number of community initiatives, like conferences and the publication of the technical journal. However, the community was not fully subordinated to the CCITT, but came about through what can be described as "coordinated emergence" – partly structured, partly unorganised.

In the theoretical literature on communities and the commons it is typically held that communities must possess the ability to self-organise, and consequently to shape the supporting and institutional arrangements, to effectively govern a common resource.[136] The participants in the Chill project lacked many of these possibilities. They were bound by the formalities of the CCITT process, even though there were many ways to by-pass these. The use of delayed contributions and the organising principles of special teams and forums are good examples of such actions in the previous periods. Now, the by-pass operations were all the more evident: the Chill community organised local user groups and collaborative projects beyond the CCITT framework. Yet, the Chill community was one of the official objectives delegated to a part in the CCITT named the "Sub-Working Party XI/3-2" in the study period that ran from 1981 to 1984. The group had previously been responsible for the organising of the Team of Specialists and Implementors' Forum in the preceding two study periods. Now, the boundary-spanning entities like the Team of Specialists and the Implementors' Forum were dropped and the main Chill activities were brought into the formal hierarchy of CCITT working groups. In 1981 the agenda of the group was formulated as follows: "The objective of Sub-Working Party XI/3-2 in the 1981-1984 study periods is to encourage and facilitate the widespread use of CHILL as a standardized basis for engineering reliable software in telecommunication."[137] The subject ("engineering reliable software in telecommunications") and the intent ("facilitate the widespread use") were clear. The group was to facilitate training in Chill and promote it by presenting papers on Chill at conferences and in journals, establish a Chill users' conference, establish a Chill bulletin and investigate the possibilities of making Chill documents more easily available to a larger audience.

The core group looked fairly similar to that of previous CCITT study periods, although somewhat smaller and less active than in the implementation phase. The group was made up of some veterans, like

[136] The standard reference is Elinor Ostrom, *Governing the commons : the evolution of institutions for collective action*, The Political economy of institutions and decisions (Cambridge ; New York: Cambridge University Press, 1990).

[137] Sub-Group XI/3-2, 6-16 April 1981, "Study programme for the period 1981-1984 for CCITT Sub-working party XI/3-2 (Question 8/XI)", COM XI 25-E. COM XI 1981-1984, CCITT, ITUA

Kristen Rekdal, as well as newcomers, like the successor of Remi Bourgonjon in Philips, Kees Smedema. Organisations that had actively participated in the Team of Specialists and the Implementors' Forum, like Philips and the NTT, started out in a fresh spirit and with new participants.[138] Some of the members represented telecommunication administrations, while others came from research establishments or manufacturing firms.[139]

Several coordinated initiatives were started in the first half of the 1980s. This included the creation of the *Chill Bulletin* and the initiation of a series of Chill Conferences. The CCITT would also publish an official user's manual to the language. The manual was intended as an elementary introduction to the language, much more approachable than the official CCITT language definition and not least, compared with the mathematically rigorous approach of the "Formal definition of Chill", which finally hit the shelves in the early 1980s. Both were published as so-called CCITT manuals. Tellingly, the language was "described fairly informally using prose and a number of examples".[140]

Furthermore, a variety of tutorial sessions took place inside telecommunication companies, schools of engineers, universities and professional societies. The sessions entailed the production of educational materials such as slides and exercises, material that would circulate among participants of the community afterwards. The CCITT summed up the activities of the Chill community in May 1984 and produced the following data about their extent and frequency:

---

[138] Remi Bourgonjon of Philips, who had led both of the special task forces from 1975 until 1980, was succeeded by Kees Smedema from 1981. Smedema was active in the CCITT until 1984. The NTT was represented by Norio Sato, who succeeded Katsumi Maruyama. The ITT was, as in previous periods, represented by various people.

[139] The available documentation on what went on within the CCITT in the study period from 1981 to 1984 is a lot scarcer than in previous periods. The amount of information found in the private archive of Kristen Rekdal is sporadic and less systematic than those for previous periods. Full details are only available on the meetings held up until early 1982, including contributed documents and lists of participants. The limited amount of material found in the ITU archives in Geneva is comparable to that of previous study periods, which means no delayed contributions or temporary documents. As such, analysis that necessitates details on participation and document contribution, like the social network analysis undertaken in chapters four and five, is not possible for the period analysed here. However, some of the events were reported in the Chill Bulletin, which also published the official reports to the CCITT.

[140] CCITT, *Introduction to CHILL – The CCITT High Level Language* (1980), 2.

| Year | Press releases | Conferences, seminars, tutorial sessions | Total number of participants |
|---|---|---|---|
| 1982 | 10 | 20 | 945 |
| 1983 | 5 | 15 | 1290 |
| 1984[141] | 1 | 2 | 210 |
| Total | 16 | 37 | 2445 |

**Table 6.1: The frequency and extent of Chill-related activities, 1982-1984.**

These numbers included everything big and small, but still give a good indication of what appeared to be a rising interest in Chill in the first half of the 1980s. The number for 1984 includes only activities for the first quarter of that year, and there are reasons to believe that the numbers for the year in full would match those of the two previous years. Such a trend is comparable with a number of similar indices, like the number of subscribers to the *Chill Bulletin* and the participation in the Chill conferences. The *Chill Bulletin* was issued between September 1981 and May 1984. It increased its circulation from about 180 in 1981 to about 400 subscribers in 1984.[142] Five Chill conferences were held during the 1980s and one in 1990. The first was held in Lyngby, Denmark in 1981. The second was held in Lisle, Illinois in 1983, the third in Cambridge in 1984, and the last two in Munich and Rio de Janeiro in 1986 and 1990, respectively.[143] The number of active participants and an estimate of general participation are given in the table below.

[141] This number contains only information about the first quarter of 1984, due to the sources used.

[142] Information about these aspects is found in "Reply to Question 8/XI - Maintenance, training, compliance and environment aspects of CHILL", COM XI 1981 - 1984, CCITT, ITUA.

[143] The conference proceedings utilised here were found in KRC. Some of the proceedings can also be located in various libraries. No information has been retrieved about the first conference, held in Lyngby, Denmark.

| | 1983 | 1984 | 1986 | 1990 |
|---|---|---|---|---|
| Papers | 32 | 33 | 31 | 42 |
| Authors and co-authors | 49 | 47 | 60 | 99 |
| Contributing organisations | 19 | 22 | 19 | 22 |
| Estimated number of participants | 70 | 130 | 140 | Unknown |

**Table 6.2 The Chill conferences.**[144]

The Chill conferences were initiated by the CCITT, but organised by some of the most active users of the programming language, like the ITT, AT&T (after the joint venture with Philips), Siemens and Telebras. As the table above reveals, the conference grew in scale, although the number of active organisations represented by contributing authors or co-authors seems stable throughout the whole period. Furthermore, the conference programming seems fairly stable for the three first conferences, with a similar amount of papers and sessions.

The organisations that were active in the CCITT in the 1970s came to dominate the Chill conferences. The ITT, Philips and Siemens were the only organisations being present with papers at all four conferences, although a number of organisations took part in all the conferences through direct participation or by acting as session chairmen throughout the period.[145] Some of the organisations that were active in the two former periods of the Chill projects did not participate actively in any way on the four conferences: most notable was the absence of L. M. Ericsson, which was not very surprising given the defection of L. M. Ericsson and the Swedish administration from the Chill cause early in the 1980s.

The conferences were dominated by manufacturing organisations. The only administration with a large presence at the conference was the Brazilian Telebras. A third group of participating organisations were scientific organisations or research establishments, like CSELT of Italy and the South Korean ETRI. These organisations typically held tight links to their respective administrations. Small start-up companies like the Norwegian Urd were also active at the conferences, which is something I will return to in the next chapter.

[144] Overview of papers presented at the Chill Conference, number of authors and estimated number of participants, 1983 – 1990. Sources: Conference proceedings. The estimated numbers of participants are drawn from "Reply to Question 8/XI, 1984", and list of participants for the Fourth Chill Conference, 1986, in the KRC.

[145] The KRC contains a participant list for the fourth Chill conference in Munich in 1986. 142 participants from 29 different countries are listed, with 60 authors or co-authors being present.

| Affiliation | Country | 1983 | 1984 | 1986 | 1990 |
|---|---|---|---|---|---|
| Bell Labs (prior to AT&T Philips) | USA | 3 | 1 | - | - |
| CpqD – Telebras | Brazil | - | - | 3 | 17 |
| CSELT | Italy | 2 | 5 | - | - |
| Dansk Datamatik Center | Denmark | 3 | 2 | - | - |
| ETRI | South Korea | - | - | 2 | 11 |
| GEC Telecommunications | UK | - | - | 1 | 2 |
| GTE | USA | 2 | 3 | - | - |
| Hasler | Switzerland | - | 1 | 3 | - |
| ITALTEL | Italy | - | - | 6 | 3 |
| ITT / Alcatel | UK, USA, Belgium, Norway, Austria, France | 10 | 11 | 2 | 7 |
| Nanjing Communications Engineering Institute | China | - | - | 4 | 4 |
| NTT | Japan | 3 | 2 | 3 | 2 |
| Philips (including AT&T from 1986) | The Netherlands, Belgium, Germany | 7 | 5 | 4 | 3 |
| Telecommunication administration | The Netherlands | 1 | - | 1 | - |
| Runit | Norway | 3 | - | - | 2 |
| Scandpower | Norway | 1 | 1 | - | - |
| Siemens | Germany, Austria | 2 | 4 | 7 | 10 |
| Technical University of Warsaw | Poland | - | - | 6 | 6 |
| Telecommunication Research Lab | Denmark | 1 | 2 | - | 2 |
| Telletra | Italy | - | - | 3 | 2 |
| Universidade Técnica de Lisboa | Portugal | - | - | - | 2 |
| University of Berne | Switzerland | - | 1 | 1 | - |
| URD / Kvatro | Norway | - | - | 2 | 2 |

**Table 6.3: Number of Authors and co-authors at Chill conferences, 1983 – 1990 (ordered alphabetically).**[146]

[146] Organisations included in the table provided authors or co-authors at more than one conference. The grouping of parent organisations is as follows. For the ITT, Philips and Siemens, authors are ascribed to their parent organisation regardless of their personal host country. In the case of Philips, this includes authors from Philips Telecommunication Industry, the joint venture of AT&T and Philips, the Belgian MBLE and the German Philips Kommunikationsindustrie. For the ITT, the numbers for 1990 include authors affiliated to Alcatel CIT, after Alcatel acquired ITT's European telecommunication operations in 1986. Before that, the ITT numbers also included authors affiliated to the Norwegian ITT subsidiary Standard Telefon og kabelfabrikk (STK) and Bell Telephone Manufacturing (BTM), Antwerp, the Belgian subsidiary of the ITT. The numbers include keynote speeches.

The community of Chill designers, developers and users grew throughout the 1980s. Its growth reflected the use of the language within a few large industrial firms and the ambiguity and negativity shown by a large number of administrations. While some of the veterans of the design and implementation period stayed on, new recruits were also won. In particular, participants from Asian and South American countries came to dominate the community at the end of the decade. Some of the old hats retired from the community or went on to greener pastures during the early 1980s. One example was the lead designer of the language, Remi Bourgonjon, who bid farewell to the community with his keynote speech at the Chill conference in 1983.[147] Here, he summarised his experiences, but also looked forward and in particular described a vision of what would be the important steps to tackle "the software crisis". As discussed in chapter two, the software crisis had been an all-inclusive tag used to describe the problems associated with the software practice since the late 1960s. To Bourgonjon, Chill was only a partial solution:

> There are several approaches to tackle the software problem. High-level programming languages such as CHILL form one such approach. Although the most established approach, it still has to mature to exploit its full power. The CHILL level of programming languages is not expected to be surpassed by new techniques, at least not for many years. Programming environment, incorporating CHILL, will become operational and will further contribute to increased software productivity. The problems here lie not in the interface to the programming language but with the interface to the target system. Most new results from research are to be expected in the area of specification and design techniques. They can be a big step in reduction and mastery of software complexity.[148]

In his "thanks and farewell" speech, Bourgonjon pointed out two areas where new developments would make large contributions to software development: programming environments and most importantly, specification and design techniques. In particular, he betted on the advances in specification and design techniques, implying techniques for describing the properties of a telecommunication system and the structure of its implementation. [149] Implicitly, this meant that huge improvements in programming language design were a thing of the past.

---

[147] Remi Bourgonjon, "Programming languages, environments and Chill", *Chill Bulletin* 3, no. 1 (1983), 3 – 8.
[148] Ibid.
[149] Bourgonjon would go on to manage large software projects within Philips and claimed he did not follow the fortunes of Chill after the mid-1980s. Remi Bourgonjon, interview with author, 16 January 2009.

Did this implicitly point towards a change in the type of knowledge that was circulated in the Chill community at the time, or was this particular language-oriented community confined to incremental changes in language technology – and thus, already passé? According to Bourgonjon, he had by now started to consider Chill, and programming languages in general, in a different light. "I started to realise that the choice of programming languages, or their design, was not that relevant as when we designed Chill," he told me.[150]

## The circulation of knowledge

The proceedings of the five Chill conferences held during the 1980s and in 1990 give good indications of what type of information and knowledge was circulated and distributed in the Chill community. The published bulletin adds to this material. In the following, I analyse this material to characterise the effectiveness of the Chill community and to capture whether the gradual withdrawal of the administrations somewhat changed the dynamics of the community.

A thorough content analysis of all 138 conference papers and 30 authored bulletin contributions and informational pieces would be difficult and time consuming. Consequently, I have applied a simple classification scheme to reveal some of the characteristics of the knowledge that circulated in the Chill community at the time. Of central interest is to what extent *prescriptive knowledge* about how to use the programming language was circulated, or whether the information circulated was purely about language extensions and changes in the language design.[151] A second issue is the one raised by Bourgonjon in the quote above: whether the community was able to extend itself in terms of its subject matter, and whether important fields of knowledge about specification techniques and programming environments were "allowed" to circulate within the language-specific community of Chill practitioners.

Two classification schemes have been applied to answer the questions. The main classification scheme tries to determine what type of knowledge the articles contain, while the second tries to single out articles concerned with specification techniques and programming environments, the very same categories that Bourgonjon singled out in his article from 1983.

The main classification scheme is made up of three main article types. Firstly, articles presenting *prescriptive knowledge* about how to use the programming language, or reports on the experience of use of the

[150] Remi Bourgonjon, telephone interview with author, 17 March 2011.

[151] On the terminology of prescriptive and propositional knowledge, see Joel Mokyr, *The gifts of Athena : historical origins of the knowledge economy* (Princeton, [N.J.]: Princeton University Press, 2002).

programming language containing real-world examples, either in natural language or in code, are singled out. This group includes articles including real code examples and illustrations of use. This is knowledge that typically is described as “knowing how”. Furthermore, purely informational articles, either concerning coming or past events, implementations or related technical projects are grouped together. The last group of articles concerns language development and language specific concepts and features, either as formal propositions or information about extensions, modifications or propositions to either the recommendation or a subset of the language. This category might look superficial or overlapping with the two broader categories, as propositions to change the programming language could be envisioned as some sort of prescriptive knowledge. However, the articles in question are so distinct and mainly about programming language design rather than prescriptive knowledge about language implementation or application or systems programming that singling them out is warranted. This type of knowledge is propositional, in a way they are of a “knowing why” kind.

The scheme applied consists of mutually exclusive categories, which means that an article can either contain prescriptive knowledge or information. The main demarcation line has been what is perceived as the major part or point in the article at hand, or what is presented as such in its abstract. If the content in general is prescriptive in nature, but also includes specific informational aspect about the implementation or technique, I have categorised it as prescriptive. The other way around, if the prescriptive elements are minor when compared with the larger parts of the article, it is categorised accordingly. Some articles are, however, not possible to categorise within this scheme. Typical examples are broad discussions about standardisation or programming at large. Such general addresses have been left unclassified. This also goes for articles that are impenetrable and incomprehensible.

It is not straightforward to apply such a classification scheme to a wide range of rather esoteric articles. Naturally, this exercise is hinged on subjective constructed criteria and my own interpretation of the content of the articles in question. A typical problem involves the technical nature of many of the articles, making them hard to read and difficult to understand. Furthermore, many of the articles are complex in nature, containing both informational aspects and novel technological knowledge. As such, drawing the line between informational articles and those circulating more codified knowledge is fraught with difficulties.

The classification scheme still has possibilities that more than make up for its difficulties. First of all, it makes it possible to discuss the real content of the knowledge circulations within the Chill community somewhat more precisely. Furthermore, it makes it possible to discuss whether the bulletin and the conferences really distributed and circulated knowledge in

"codified" form, meaning rather precise and tight know-how, or only papers of an informational character.[152] Below, I present the findings after surveying all papers printed in the conference proceedings for the four conferences where I have been able to track them down, as well as from the issues of the Bulletin running from 1981 to 1984.

| | Prescriptive knowledge | Information | Language development | Unclassified |
|---|---|---|---|---|
| Bulletin | 5 (15.6 %) | 16 (50%) | 9 (28.1%) | 2 |
| Conference '83 | 10 (32.3%) | 11 (35.5%) | 8 (25.8%) | 2 |
| Conference '84 | 11 (35.5%) | 7 (22.6 %) | 10 (32.3 %) | 3 |
| Conference '86 | 16 (50%) | 11 (34.4 %) | 5 (15.6 %) | 0 |
| Conference '90 | 15 (37.5) | 18 (45 %) | 5 (12.5 %) | 2 |

**Table 6.4: Articles from the Chill Bulletin and Chill Conferences, categorised.**

Regarding the results, I will address the content of the conference proceedings first. The numbers reveal no obvious pattern dynamic and no article category dominates, except the diminishing relative numbers of articles primarily concerned with programming language issues. As revisions and additions to the programming language occurred less regularly past the third study period of Chill-related work within the CCITT, this tendency was natural. The relatively large number of articles concerned with language development or programming language issues at the conferences in 1983 and 1984 concerning language development was related to ongoing discussions in the CCITT, in particular piecewise programming and compilation.

The articles containing prescriptive knowledge in one form or another are a plenty, at all four conferences. In 1986, 50 per cent of all the papers presented were dominated by prescriptive knowledge in some way or another. Generally, it seems that these papers were mainly concerned with compilation techniques in various ways. At the 1986 conference this accounted for nine out of 16 articles. The typical issues discussed in these compilation-oriented articles are concerned with compilation design combined with the newly added piecewise compilation feature in the 1984

[152] On the codification of knowledge, see Margherita Balconi, Andrea Pozzali, and Riccardo Viale, "The 'codification debate' revisited: a conceptual framework to analyze the role of tacit knowledge in economics", *Industrial and Corporate Change* 16, no. 5 (2007); Cowan, David, and Foray, "The Explicit Economics of Knowledge Codification and Tacitness"; Björn Johnson, Edward Lorenz, and Bengt-Åke Lundvall, "Why all this fuss about codified and tacit knowledge", *Industrial and Corporate Change* 11, no. 2 (2002); Paul Nightingale, "If Nelson and Winter are only half right about tacit knowledge, which half? A Searlean critique of 'codification'", *Industrial and Corporate Change* 12, no. 2 (2003).

version of the Chill recommendation. As such, they tied in nicely with a concern that was common to both the core and periphery of the Chill community, namely how to facilitate the use of the programming language rather than its use per se. The prescriptive knowledge that circulated within the Chill community was, in general, implementation knowledge. Knowledge that could ease the implementation of the programming language in various settings was readily available, and typically it was related to issues like portability between target systems, or its close cousin, portability among host systems. Another issue that was debated was how new language features influenced compilation techniques. On the other hand, almost none of the articles contained knowledge or examples of systems programming, as those articles touching on these issues are all informational in character, with precious few including real-world examples or code.

Let us now turn to the *Chill Bulletin*. Both official reports from the CCITT and authored articles were featured in the four volumes and six numbers I looked into, but editorial content like the editor's introduction and listings of coming events are not part of the survey above. All in all, the Bulletin was dominated by official reports or output from the core Chill community within the CCITT. Draft proposals on new language features, information about CCITT meetings and so on took up a large proportion of the material in the Bulletin.

Only a handful of articles on particular aspects of the programming language can be said to contain prescriptive knowledge, and when that was the case, they were mainly concerned with implementation issues like compiler validation. All in all, the bulletin reported on the function of the Chill community rather than the knowledge held by it. The *Chill Bulletin* drew inspiration from other informal publications related to specific programming languages, like the *Algol Bulletin* and the *Simula Newsletter*.[153] Compared with the *Algol Bulletin*, which was published by the Association for Computing Machinery (ACM) from March 1959 till August 1988, the *Chill Bulletin* was a blip, a short-running publication containing little of real interest to the members of its community. Where the *Algol Bulletin* published informal papers and discussion from a wide range of authors, the contributions to the *Chill Bulletin* were limited to a few authors and the journal had to include a number of official CCITT documents to fill its pages. The *Algol Bulletin* ran for 52 issues containing high quality articles, although in a pretty informal manner quite unlike a scholarly journal.

[153] For the former, see the 52 issues of the *Algol Bulletin*, from 1959 to 1988. The *Algol Bulletin* is available in its entirety from the ACM library, see http://portal.acm.org/ For the latter, see sporadic information in Holmevik, *Educating the machine : a study in the history of computing and the construction of the SIMULA programming language*.

However, the comparison would also reveal that the *Algol Bulletin* was preoccupied with language development like propositions on changes in the programming language itself, which also were themes that made up nearly 30 per cent of the articles featured in the *Chill Bulletin*.

As a further extension to this survey, I have also identified which articles were concerned with the relationship between Chill and the two promising fields pointed out by Bourgonjon in his keynote speech at the second Chill conference: programming environments and specification and design techniques.[154] Here, the schemes are obviously not exclusive in terms of the previous three categories, and applied only to a subset of articles to the total.

| | **Specification and design techniques** | **Environments** |
|---|---|---|
| Conference '83 | 2 | 2 |
| Conference '84 | 6 | 6 |
| Conference '86 | 4 | 9 |
| Conference '90 | 2 | 12 |

**Table 6.5: Articles and papers concerning environments and specification techniques in the *Chill Bulletin* and at the Chill conferences**

Discussions about the framework in which Chill would be used, typically called the environment, started to draw attention as the programming language went into use. This triggered a series of discussions about the toolbox that the use of Chill would necessitate, especially within large organisations. This tendency is evident in the subjects tackled at the Chill conferences. Articles concerned with specification techniques were fewer. They were mainly confined to discussions about the use of the specification language SDL, the CCITT specification description language. Often, the articles were also about various environment issues, and therefore more of an indication of the perceived importance of this issue.

The prescriptions on how to compile Chill, or how to develop a compiler for Chill, which dominated the circulations of the Chill community, are examples of codified procedural knowledge. Within the literature on the economics of knowledge, it is widely held that linear procedures are quite easy to codify, and that this would lead to rapid established and codified practices. [155] Another issue is the fact that compiler techniques were perceived as something within the domain of computer science. The

[154] Remi Bourgonjon, "Programming languages, environments and Chill," *Chill Bulletin* 3, no. 1 (1983), 3 – 8.
[155] Cowan, David, and Foray, "The Explicit Economics of Knowledge Codification and Tacitness".

development of systems would be less general, and less likely to be codified and available as prescriptive knowledge within the knowledge community.

## Crossing boundaries?

Did the conferences facilitate knowledge circulation among participants that worked for different and competing organisations? Conference papers can only be that much in terms of knowledge circulation. In a practice where the actual output of work is written code, there was at least a possibility of the transmission of prescriptive knowledge through conference proceedings.

As shown above, participants from Siemens, ITT and Philips continued to be present at the conferences, and contributed a substantial share of the papers. To some extent, they continued to influence the technical changes made to the language through the CCITT. However, to what extent were they interested in being open about implementation, specification and systems programming within the wider community present at the conferences? Let me briefly consider the content the papers presented by participants from Siemens at the conferences in 1983, 1984, 1986 and 1990, as an example. As one of the largest, earliest and most successful manufacturers using Chill, participants from Siemens were always present at the conferences. However, the firm was never a huge presence at the conferences, as they presented just a modest number of papers at each of the four conferences.

| | **1983** | **1984** | **1986** | **1990** |
|---|---|---|---|---|
| Papers in total | 32 | 33 | 31 | 42 |
| Siemens | 2 | 4 | 5 | 4 |

**Table 6.6: Siemens at the Chill conferences**

In 1983, Siemens was present with two authors, presenting papers on what were the two most important advances in the next version of Chill language: piecewise compilations and the input-output facilities in the language. Both dealt with what were, at the time, concerns related to programming language design, and reported on practical experiences from the Chill use in the EWSD system development. In particular, the paper on piecewise compilation involved an exposition on how the issue of separate compilation had been dealt with before the programming language itself had any strict rules about the way to make this possible.[156]

In 1984, of the four papers presented by Siemens employees, one was still concerned with language design, and another followed up on their

[156] Reithmaier, "Compilation Control in a Large CHILL Application"; Winkler, "A New Methodology for I/O and its Application in CHILL".

interest in input-output facilities by reporting on how to implement the newly established rules in the 1984 version of the Z.200 language definition.[157] One was a proposition of how to integrate various tools into a coherent programming environment or tool chain and the last paper reported on the experiences of using Chill in an experimental switch called Bigfon.[158]

The 1986 conference was organised by Siemens in Munich and one common thread bound together the five Siemens papers: the portability of software written for one line of switches (processors) to another.[159] The background to this was the introduction of new types of processors into the EWSD lines of switches, the Motorola MC68020 processors. This necessitated, at least, the recompilation of the software, and put one of the main design criteria of Chill to test: the ability to write programs in a "machine independent manner".[160] One of the papers was about a specific compilation technique for a new processor Siemens was about to use in their EWSD systems, making explicit implementation knowledge accessible.[161] Others were partly about language design, partly about other implementation issues.[162]

In 1990, the 10-man delegation of authors from Siemens presented four papers in total. One of the papers was again concerned with

---

[157] Winkler, "The Realization of Data Abstractions in CHILL"; T. Mehner and J. F. H. Winkler, "An Implementation of the New CHILL-I/O", in *Third CHILL Conference* (Cambridge University: ITT Europe, 1984).

[158] Peter Meyer, "A CHILL-based Systems Development for BIGFON", in *Third CHILL Conference* (Cambridge University: ITT Europe, 1984); T. Mehner, R. Tobiasch, and J. F. H. Winkler, "A Proposal for an Integrated Programming Environment for CHILL", in *Third CHILL Conference* (Cambridge University: ITT Europe, 1984).

[159] NM. Clark, K. Neuhaus, and G. Walter, "Support Software Environment for a Multi-Processor-Development", in *Fourth CHILL Conference* (Munich: Simenes AG, 1986); H. Hey and K. Neuhaus, "CHILL Semaphore technique for Multiprocessing", in *Fourth CHILL Conference* (Munich: Simenes AG, 1996); J. Holden and A. Pink, "A Globally Optimizing CHILL Code Generator for the Motorola MC68020", in *Fourth CHILL Conference* (Munich: Simenes AG, 1986); Peter Meyer, "Process Communication in a CHILL Environment", in *Fourth CHILL Conference* (Munich: Simenes AG, 1986); M. Clark and G. Walter, "CHILL Language Solutions for Mixed Data Formats", in *Fourth CHILL Conference* (Munich: Simenes AG, 1996).

[160] *CHILL - The ITU-T Programming Language*, ITU-T Recommendation Z.200 (1999), 1.

[161] Holden and Pink, "A Globally Optimizing CHILL Code Generator for the Motorola MC68020".

[162] In particular Hey and Neuhaus, "CHILL Semaphore technique for Multiprocessing"; Clark and Walter, "CHILL Language Solutions for Mixed Data Formats".

programming language design, this time on object-oriented aspects.[163] One was concerned with compilation design; another more broadly on compilation techniques and the last one was about the use of the specification language SDL together with Chill, reinforcing the impression that the papers stemming from Siemens were largely concerned with implementation issues.[164] Some of these papers were fairly explicit and propositional in character, opening up to the circulation about, largely, new compilation techniques, which was something held in common by many of the papers presented by other participants from other manufacturing firms. It seems that many of the ideas in the papers by Siemens participants travelled and were picked up by others. The papers on piecewise compilation techniques are the most prominent example of this.

The community formed around the programming language Chill was, to some extent, able to circulate and create knowledge, both within and outside the formal boundaries of the CCITT as well as across firm boundaries. Where the CCITT meetings were focused on standardisation and language development, the conferences reported on real implementations and systems programming. However, the function of the community was also limited: one indication of this was the strong focus on implementation issues rather than application or systems issues. Furthermore, while knowledge of a prescriptive type was circulated to a certain extent, it seems that the effect of this circulation was fairly limited: few new organisations that were "recruited" to the Chill community would apply the programming language to large-scale switching development projects, and even fewer would circulate knowledge about such projects within the community. With the telecommunication administrations leaving much of the community to its own devices, the kind of knowledge that was shared was strongly related to the large industrial firms that were actively using Chill.

## Some conclusions

This chapter has explored the general lack of interest in Chill shown by many telecommunication administrations in the 1980s, the development of the technical community of Chill designers, users and developers and the

[163] Diebl, Schulz, and Winkler, "Object-CHILL: The Road to Object Oriented Programming with CHILL".

[164] J. Schefer, J. Schiffer, and J. Weiser, "A Machine Independent Model for Flexible Construction of CHILL Code Generators", in *Fifth CHILL Conference* (Rio de Janeiro: Telebras, 1990); A. Pink, "Fault Correction in a Running CHILL System", in *Fifth CHILL Conference* (Rio de Janeiro: Telebras, 1990); G. A. Schlaffke, J. Lantermann, and G. Becker, "A CHILL Procedure Tracer For a Real Time Multiprocessor Environment", in *Fifth CHILL Conference* (Rio de Janeiro: Telebras, 1990).

subsequent technical improvements made to Chill after its ratification as an official CCITT recommendation in 1980. All in all, this has revealed a rather paradoxical setting: the CCITT, the organisation that had been described as "the anchor of a regime that facilitated bilateral monopolistic bargains, reinforced national monopolies, and limited the rights of private firms in the global market", facilitated the creation and subsequent improvements to a tool that became all the more controlled by a set of manufacturers striving for new international markets. [165] The limited appeal to the telecommunication administrations, which at the outset had seen Chill as a strategically and technologically important project, was most discouraging. When even the Norwegian administration did not find it important to demand Chill when purchasing new digital switches – it seemed highly unlikely that anyone else would do so. In this regard, the use of Chill within administrations was a real failure. The main initiators of the Chill project were the first to withdraw from it. Furthermore, as the technological practitioners started to realise that the choice of programming language was perhaps not the most important issue through which to express their development virtues, the impetus behind Chill started to seem futile.

The pattern of rejection and adoption among administrations were clearly bound up in the shifting strategies of the division of programming labour among administrations and manufacturers throughout the 1980s. The prospective status of a Lingua Franca was more likely when the administrations moved towards technical independence by moving programming within their own realms, while Chill became an obscure dialect as soon as the administrations left much of the technical development to the manufacturers.

In terms of development and refinements, the early 1980s were a period where closer interaction with users fed back into improvements to the language design, and in particular improved the few software engineering elements that existed in the language. These changes were considerably less difficult to get approved in the CCITT working group than what had been the case under the modus operandi in the two previous study periods of CCITT work. This hinged on a change in the decision-making structure within the core Chill community, which was less tense and conflicting than in previous periods, as the group was "united" around a real object: the Chill recommendation. As such, while the prospects of a functioning and effective core Chill community were quite meagre at the end of the Implementation Forum period, the prospects of the Chill community seemed considerably more positive in the mid-1980s.

---

[165] Cowhey, "The international telecommunications regime: the political roots of regimes for high technology": 176.

The Chill community expanded during the first few years of the 1980s as it drew interest from countries that had been peripheral at best during the inception years, like China, India and Brazil. However, few new manufacturing firms participated in the community, as it was still dominated by firms like the ITT, Philips and Siemens. As will be made clear in the coming chapter, where I analyse the use of Chill in large organisations, this was all due to Chill's limited appeal to existing manufacturers that had already come a long way in applying other programming languages to their development programs.

# 7. Possibilities and opportunities: entering markets with Chill

The defection of the national administrations from the Chill cause coincided with rapidly changing conditions for many telecommunication equipment manufacturers. The ties between what often had been nationally confined manufacturers and administrations fell apart and trade in telecommunication equipment surged internationally, in particular in the second half of the 1980s. Previously closed markets were opened up to outside contenders as "national champions" lost their footing and home base.[166] Paradoxically, this could have put Chill in the ascendant once again. However, this time around the diffusion had to involve independent tool vendors that catered for the users of the programming language rather than the established manufacturers.

This chapter analyses the possibilities for new Chill use that emerged in the second half of the 1980s and tries to answer why particular entrepreneurial firms were created to seize these opportunities. In particular, I focus on the ideas and knowledge that were commercialised. I try to answer why some ideas were brought to the market by new and entrepreneurial firms, while others would depend on already established firms and still others would never reach outside large manufacturers.[167] I focus on the possibility to commercialise products related to Chill programming. In addition to this, I explore how these possibilities were pursued and to what extent the ventures were successful or not. I focus in particular on the commercialisation of the Nordic compiler project through the firm Urd, the most extensive entrepreneurial effort related to Chill. In the end, I conclude on whether the extensive re-regulation and liberalisation of the telecommunication industry from the second half of the 1980s represented a new possibility for further diffusion of Chill or whether it was a set of rather unfeasible opportunities. This includes a review of the fate of Chill throughout the 1990s until the last maintained recommendation published by the ITU in 1999.

## The "ancien regime" and the new beginning

By the late 1980s Chill's status as a standard was severely weakened. The faltering position of formal international agencies of the "ancien regime,"

[166] An overview of the changes in the trade of telecommunication equipment of the time is OECD, "Telecommunications Equipment: Changing Markets and Trade Structures, No. 2", in *OECD Digital Economy Papers* (OECD Publishing, 1991).

[167] This question mirrors the general concern raised in Nicholas Dew, S. Ramakrishna Velamuri, and Sankaran Venkataraman, "Dispersed knowledge and an entrepreneurial theory of the firm", *Journal of Business Venturing* 19(2004).

like the CCITT, followed the liberalisation of the telecommunication administrations. This made the possibility of Chill as a mandatory requirement seem all the more unlikely.[168] However, as established manufacturers started to target new markets and start-ups tried to wrestle their way into the rough seas of the international telecommunication equipment market, Chill could play a new role. To compete in the international markets, new demands were put on the equipment manufacturers, demands that could be met by developing software more effectively. Chill was not without technical merit and in the mid-1980s it was more capable then before. Tools were made available to supplement the language and suddenly Chill looked like an attractive offer again, despite its faltering status as a standard approved by the CCITT. These tools were products, in contrast to the early example of the ITT's hiring of the firm Massachusetts Computer Associates, an outside software contractor, to create a compiler for the System 12 switch. ITT's further reliance on outside contractors for new developments and maintenance of their Chill tools, like the compiler development with Intermetrics and the services and further developments made by Richard Daley Associates, followed this model of using sub-contractors rather than products bought through market exchange.[169]

In a paradoxical way, the technical merits of Chill were reinvigorated and strengthened just as the ties between governing agencies and the manufacturers loosened, and the possibility to release software tools as products to the market was emerging. As the ties between manufacturers and administrations were severed, firms that had previously been limited to operations in their national market tried to establish new ties, but of a different kind: strategic alliances, mergers, acquisitions and eventually the increasing use of outside suppliers became increasingly common throughout the decade. The joint ventures by ITT and Alcatel and by the AT&T and Philips are important examples, and by the late 1980s, these were joined by the merger of the two British manufacturing firms, the telecommunication division of General Electric Company (GEC) and Plessey, an organisational entity that eventually would end up in the joint acquisition of Plessey by

[168] The concept of an "ancien regime" in telecommunications, and how the ITU supported this regime, was explained in chapters one and two. See also Drake, "The Rise and Decline of the International Telecommunications Regime".
[169] Richard Daley, emails to author, April 2011.

GEC and Siemens in 1989.[170] The number of similar large-scale reorganisations in the telecommunication equipment industry only accelerated in the 1990s.[171] These reorganisations created an opening for outside suppliers of software tools and programming services. New businesses could target the telecommunication equipment industry by selling compilers and other programming tools as products to the industry. Before this, these products had been made by the large manufacturers that had already committed themselves to Chill and to research establishments with strong ties to the telecommunication administrations.[172] The early examples of reliance on outside aid have already been mentioned, where the most international of all the manufacturing firms, the ITT, contracted out the development of a new compiler for Chill to the specialist firm Massachusetts Computer Associates in the early 1980s.[173] However, this early example was one of contracting, as the compiler was never traded openly in a market, but was limited to the one customer, the ITT.

## New ventures and new environments

A programming language and its accompanying set of compilers are only part of what is needed to develop software. The programming language is only a set of rules for writing programs and the compiler is what transforms this writing into executable computations. Between the process of writing the code and handing it over to the compiler, other tools can be applied to ease the work associated with software development. This part of the tool chain, including the compiler, has often been called the *software*

[170] On the considerable differences between the equipment markets in various Europe countries, see Enzo Pontarollo, "Procurement and market structure in the telecommunications industry : A European survey", *European Journal of Purchasing & Supply Management* 1, no. 2 (1994). On the reorganisation of GEC and Plessey, see Owen, *From Empire to Europe*, 282-88.

[171] Fernand Amesse et al., "The telecommunications equipment industry in the 1990s: from alliances to mergers and acquisitions", *Technovation* 24, no. 11 (2004).

[172] As noted in chapter one, the similarity to the changes in the production of machine tools in America, which were first made on an ad hoc basis by their users and later on spawned the emergence of firms devoted to machine production from about 1840 to 1880, is striking. See Rosenberg, "Technological Change in the Machine Tool Industry, 1840 - 1910".

[173] See chapter five.

*development environment*.[174]

Some elements of these environments could be language independent, making way for generic tools. One example is sophisticated code editors, the word processors of software development if you like. Other elements would, to some extent, have to integrate the programming languages into their midst to be effective, like the compiler. In addition to the compiler – which forms an important element in any programming environment - these tools could be so-called debuggers, which is a type of software used to find the cause of an error (a so-called bug) that exists in a program. Other valuable tools in such environments could be linkers, which tied together separately compiled modules. Many other types of tools were available to the software developer in the 1980s, a point in time that generally marked a shift in focus from a past centred on the programming language.

Programming languages for telecommunication were something special, and accordingly, the elements in the software development environments were peculiar to the telecommunication industry. Furthermore, as Chill was a very peculiar programming language, so the environments would have to be rather language-centred. According to Remi Bourgonjon, what such environments should consist of was also a difficult question:

> In large system developments, such as SPC switching systems, it is [....] difficult to decide what should be in the programming environment. For example, in SPC systems development almost everything is related to everything else. The compiler has a strong relation with the debugger, which has a relation the operating system, file system, telephony test system etc. The telephony test system has a relationship to the telephony programs etc.[175]

Accordingly, the environments used in telecommunication programming were often tightly integrated with the telephony systems, something that gave those directly involved with such systems a head start. Large firms using Chill, like the ITT, Siemens and Philips, had already in the early 1980s developed a number of tools to strengthen and complete their own software

[174] The terminology of software development environments was, to some extent, a thing of the 1980s, where the term proliferated in conference proceedings, like those from the International Conference on Software Engineering, and in journals like the *Software Engineering Journal*. As such, the term is historically correct when discussing the period under consideration, although it might not be in vogue at the moment. A similar term, so-called programming environments, appeared earlier, in the late 1970s, and implied much of the same thing.

[175] Here, Bourgonjon used the terminology of programming environments in place of software development environments. See Remi H. Borugonjon, "Programming Languages, Environments and Chill", *Chill Bulletin* 3, no. 1 (1983).

development environments.[176]

The entrepreneurial efforts to commercialise knowledge gained through the Chill project were geared towards the needs of strengthening such development environments, meaning that they targeted openings in the environments of established firms. At first, the primary focus was the development of compiler technology, which is the most language-specific part of a development environment, but other tools were also made available through entrepreneurial firms. Consequently, they targeted a market where in-house developments already had been going on for a number of years. However, the entrepreneurial ventures were not without their own history. They were, in some way or another, related to already existing Chill users or researchers.

Three "and a half" ventures tried to commercialise products or services related to Chill through market exchange, meaning that the product or service was made available from more than one contracting customer.[177] All the ventures started out by selling compilers and providing services to users of compilers, although Urd would eventually branch out into a series of different implementation tools, eventually ending up offering a complete software development environment. The ventures under consideration in this chapter were primarily involved in the introduction of tools to the market, but I do also briefly consider some additional service offers that were made by firms not offering products. Only one of these ventures relied completely on Chill-oriented products.

The first and largest venture to introduce products that could assist in using the Chill programming language was first introduced to the market by Urd Information Technologies. Urd Information Technologies was established in Trondheim, Norway, as a direct result of the Nordic Chill Compiler project. It was established in 1984. It was started by Kristen Rekdal, who had held prominent positions within the CCITT Chill project for a long time, and was set up to commercialise Chipsy (an acronym for Chill Programming System), the programming environment that had been developed by Runit for the Nordic telecommunication administrations since 1978. Starting out as a compiler for a limited number of target hardware platforms, the programming system now included other tools, such as testing

---

[176] See for example Ibid; Clark, Neuhaus, and Walter, "Support Software Environment for a Multi-Processor-Development"; Mehner, Tobiasch, and Winkler, "A Proposal for an Integrated Programming Environment for CHILL"; Meyer, "Process Communication in a CHILL Environment".

[177] On the difference between software contractors and products, see Campbell-Kelly, *From airline reservations to Sonic the Hedgehog : a history of the software industry*; Campbell-Kelly and Garcia-Swartz, "From Products to Services: The Software Industry in the Internet Era".

tools and tools for simple configuration management, meaning the task of tracking and controlling changes in the software. In short, it gradually filled in the space in a Chill-oriented software development environment.[178]

The second product that was tried and sold through a market was a Chill compiler by the British company Imperial Software Technologies, which had been funded in 1982 as a spin-off from the Imperial College of Science and Technology in London. Imperial's Chill compiler was, however, not a result of work done at the Imperial College, but a product developed through a Danish-British relationship. The British telecommunication manufacturer GEC, the Dansk Datamatik Center (DDC), which was founded by Dines Bjørner in 1979 as a sideline of the technical university in Lyngby, and the Danish Telecommunication Research Laboratory extended the formal approach to compiler design that Bjørner had pioneered throughout the 1970s, by developing a Chill compiler.[179] The main architecture of the GEC-commissioned compiler stemmed from this work, while the code generator was eventually coded by Imperial Software Technology. The project also involved the use of a Chill compiler in a proposed Ada programming environment, making it possible to combine programming projects that used both languages. GEC was one of the main participants in this project, which was sponsored by the EEC.[180]

This compiler was put on the market by Imperial Software from around 1986, making the technology available to customers other than GEC.[181] The main person behind the work at Imperial Software was Peter J. Smith, who had previously worked for the ITT creating a compiler for their System 12 switch. Smith had also been active in the CCITT on behalf of the ITT in the study period from 1981 to 1984, contributing a number of papers on the revisions of Chill.[182] By the time the GEC compiler was put to use, the company was locked into a dogfight with the second British

---

[178] On Urd and Chipsy, see various documents available in the NTR archives, which contain extensive material on Urd and Chipsy. See in particular boxes "L 0135 Samarbeid" and "L 0136", series "Da, 1961 – 1996," NTR.

[179] Peter Haff and Søren Prehn, "The TFL/DCC CHILL System Development", in *Second CHILL Conference* (Lisle, Illinois: Bell Laboratories, 1983).

[180] Meiling and Palm, "A Comparative Study of CHILL and Ada on the Basis of Denotational Descriptions".

[181] Peter J. Smith, "Experiences in Achiving A Full Implementation of CHILL", in *Fourth CHILL Conference* (Munich: Siemens AG, 1986); D. R. Johnson and C. P. Miller, "Testing a CHILL Compiler", in *Fourth CHILL Conference* (Munich: Siemens AG, 1986). See also Kristen Rekdal, "Report from forth Chill conference, Munich, 29/9 1986 – 2/10 1986", 8 October 1986, box "L 0136 Samarbeid", series "Da, 1961 – 1996", NTR.

[182] Details given in the biographies section of the conference proceedings of the third CHILL Conference (Cambridge: ITT, 1984).

manufacturer, Plessey, and the System X switching system, which had previously been a joint development programme between the two companies and the British telecommunication administration, was in disarray. The now privatised telecommunication division of the Post Office, now named British Telecom, bought a substantial number of System X switches, but by 1985, Ericsson was let into the British network. By the late 1980s, GEC, together with Siemens, bought the remains of Plessey. System X, on the other hand, did not experience any major success, as its diffusion was largely confined to the British Isles. The Chill compiler did not, however, survive the constant upheavals at GEC in the late 1980s, or at Imperial Software.[183]

The "half" in the "three-and-a-half" was the Dutch telecommunication administration, which sold their Chill compiler on ad hoc basis.[184] The compiler had been developed throughout the period of the Implementors' Forum, where the Dutch PTT was one of the most active participants.[185] In the 1980s, this compiler was developed into a software development system, consisting of a set of tools such as compilers for different target computers, a formatting tool for the source code and a debugger.[186] The effort to sell the compiler or the development environment did not develop beyond the preliminary and ad hoc basis.

In the 1990s, the American company Cygnus pioneered the open source business model by providing software developer tools as open source software and delivering services to their users.[187] A Chill compiler was developed by Cygnus in the early 1990s, and it was later included in what became the GNU Compiler Collection (GCC), an open source collection of

---

[183] Information about GEC and System X is briefly given in David Parker, *The Official History of Privatisation Vol. I: The formative years 1970-1987* (Routledge, 2009), 260. See also Owen, *From Empire to Europe*, 282-88.

[184] Details about this are found in Kristen Rekdal, "Report from fourth Chill conference, Munich, 29/9 1986 – 2/10 1986", 8 October 1986, box "L 0136 Samarbeid", series "Da, 1961 – 1996", NTR.

[185] On the technical background of this project, see G. H. te Sligte, "A cross-implementation of Chill on existing hardware under a commercial operating system", in *Software Engineering for Telecommunciation Switching Systems* (University of Warwick, Coventry: Institution of Electrical Engineers, 1981); Meijer and Sligte, "Status report of CCITT HLL implementation at the Dr Neher Laboratory of the Netherlands PTT".

[186] G. H. te Sligte, "A programming environment for Chill", in *Second CHILL Conference* (Lisle, Illinois: Bell Laboratories, 1983).

[187] On Cygnus, see Michael Tiemann, "The future of Cygnus Solutions : and entrepreneur's account", in *Open sources : voices from the open source revolution*, ed. Chris DiBona, Sam Ockman, and Mark Stone (Beijing ; Sebastopol, CA: O'Reilly, 1999).

compilers for various programming languages.[188] This made the core of any Chill implementations available for free. Nevertheless, the Chill part of Cygnus was never extensive: quite the contrary. By the late 1990s, The GNU Chill implementation was no longer being actively developed. Cygnus had at that point only one customer for whom they were maintaining the Chill compiler. Subsequently, the Chill compiler was pulled back from the GCC collection due to little interest.[189] The Cygnus venture, although intimately tied in with the availability of a free and open source version of a Chill compiler, points towards the possibility of other service-oriented Chill ventures. Such cases would be much harder to track down, in particular if they were small and mainly dealing with one or a very limited set of customers. Following this, and the fact that the Cygnus effort was developed quite some time after the others, I will in this chapter focus on the efforts to commercialise Chill tools in the latter half of the 1980s. Another case that I do not consider any further was the compiler developed by the NTT, which had made their Chill compiler available to their favoured cooperative manufacturers, NEC, Hitachi, Fujitsu and Oki. I consider this more in line with the old telecommunication regime and not as a result of new opportunities and market exchange.[190]

## Entrepreneurship, communities and knowledge

As seen above, the Chill community was not exactly bursting with entrepreneurship. The few examples that existed were limited. The entrepreneurial activities were initiated by or associated with people who held central positions within the Chill community and concentrated on implementation technologies like compilers and other parts of the development environments.

The three main efforts to sell Chill-related products originated with people with central positions in the Implementors' Forum: Rekdal, who founded Urd, Bjørner, who was central in the Danish compiler project that would be transferred to Imperial Software, and Meijer, who led the work of Dutch administration on Chill, were all important actors in the early development and implementation efforts of Chill. Rekdal would go on to be the chairman of the CCITT working group in the 1980-84 study period and

[188] "Chill Front End", August 29, 1998, http://gcc.gnu.org/news/chill.html (retrieved 5 April 2011).
[189] The Chill compiler was omitted from the 3.0 version of the GCC, which was released in 2001. It was removed from the GCC source tree on 15 April 2002. See http://gcc.gnu.org/news/ (retrieved 5 April 2011).
[190] As noted in chapters five and six, the details on the use of Chill in Japan are scarce and hard to come by. An overview is found in Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 425-30.

held a prominent position within the CCITT throughout the Chill project, while Bjørner, and his related research group, were active and vocal in the development of both Ada and Chill. Furthermore, Peter J. Smith, who led the Chill work at Imperial Software, had previously represented the ITT in the CCITT in the early 1980s.

Another similarity between the entrepreneurial efforts was their networked nature outside the CCITT community: Rekdal's work spanned the CCITT, the Nordic compiler project and various positions in academia. The same goes for the Imperial Software Technology project, which involved a number of companies, like GEC, and the research institute DDC and the Danish Telecommunication Research Laboratory, and implicitly, through the prior employment of Peter J. Smith, also the ITT. All in all, the early efforts to commercialise Chill tools were spun out from cooperative networks of various kinds, releasing the entrepreneurial potential of the networks established within the boundaries of the "ancien regime" of telecommunications.

However, they were also carved out by technical experts who held a rather independent position. It was no coincidence that the two main entrepreneurial efforts that I have discussed in this chapter, Urd and the Danish compiler transferred to Imperial Software, were fruits of research done at fairly independent research establishments, in countries with no dominant telecommunication manufacturer, Norway and Denmark. The central position in the Chill community held by Kristen Rekdal and Dines Bjørner was a consequence of the same free-standing positions, and would go a long way towards explaining their specific possibility to realise commercial products from their Chill activities. Furthermore, both Rekdal and Bjørner had expressed affinity for elements of formally oriented computer science. Bjørner and his partners at DDC and the Danish telecommunication research laboratory extended this towards the construction of compilers, with an explicit focus on correctness-proving of compilers aided by his formal description methods.[191]

All the cases discussed above tried to commercialise knowledge about how to implement the programming language Chill, first by offering compilers for various host computers and hardware targets – and in the case of Urd by eventually providing elements of complete software development environments. They were, in other terms, not products already available in the market. There are reasons to believe that Rekdal, Bjørner and the Dutch telecommunication administration were ahead of the curve in terms of acquiring such knowledge early on in the Chill development, providing

[191] One example is Flemming Andersen and Karsten Nyblad, "Compiler testing, theory and experiences", in *Third Chill Conference* (Cambridge University: ITT Europe, 1984).

access to advanced and comprehensive compiler solutions. In particular Bjørner and Rekdal had considerable experience in compiler design, also for other languages and towards various hardware platforms.

The community in itself could be deemed an obstacle to market creation, as the network facilitated the exchange or transfer of knowledge without the need of monetary exchange and there existed few reasons for people holding central positions within the community network to act entrepreneurially, since the knowledge available to the central actors would be much the same as to those in more peripheral positions within its borders. To people outside the Chill community, however, the knowledge was so difficult to grasp, in some ways because of the astute formalism of some of the technology, that they were not able to effectuate on it.

Compared with the continuous activities around the programming language Ada, the number of Chill-related products and services exchanged in market-like structures was meagre. As of June 1986, there were 47 compilers from 19 different vendors developed to serve the Ada market. Together with the substantial amount of other Ada-related activity within a slew of established firms, this creates the impression of an expanding market.[192] The activities in Europe expanded throughout the 1980s as well.[193]

By the late 1980s, the number of Chill compilers was not all that meagre: I accounted for 29 of them in chapter five. However, many of those were not available from vendors, but held as in-house tools by large manufacturers or created as proof-of-concepts by research establishments. A software tools exhibition was held at the 1990 Chill conference. Five organisations took part. The Warsaw University of Technology exhibited their Chillit programming environment (about which I have no further information). Kristen Rekdal showed off the Chipsy system by Urd, now named Kvatro. Karsten Nyblad of the Danish Telecommunications Research Laboratory displayed their compiler and interpreter, which had previously been developed and marketed by Imperial Software. The NTT demonstrated their software development environment and the Brazilian hosts, the CPqD, revealed a number of development tools. [194] If the exhibition was representative to what was available at the time, and there are good reasons to believe it was, it was a long way from the 19 Ada compiler vendors in 1986. Furthermore, it is not given that these exhibitors were vendors of anything, apart from the Urd/Kvatro team.

---

[192] The numbers are taken from Jean E. Sammet, "Why Ada is not just another programming language", *Commun. ACM* 29, no. 8 (1986).

[193] See for example Ada Language (Great Britain) Ltd., "Ada yearbook", (London: Chapman & Hall, 1991).

[194] The exhibition is listed in the final programme leaflet of the 5th Chill conference.

The opportunity to commercialise Chill-oriented tools rested on the opening up of established manufacturers to outside tool vendors. GEC's involvement with Imperial Software was one signal of this. Another signal was the extensive cooperation between the administration-sponsored Nordic compiler project and the ITT subsidiary STK in Norway, which was described in chapter five. Here, the key technology to the development of the so-called nodal switch was delivered by an outside party. In the same period, the ideal of Chill as something preferred by administrations was also not without merit: Siemens used Chill as a part of their campaign to get into markets like Pakistan in the second half of the 1980s. However, the signals off an opening in the market to target existing and large-scale switching manufacturers did not last. The following section looks into these dynamics in detail through a case study of the setting up and development of Urd.

## Spinning off

The most comprehensive and extensive case of Chill-related entrepreneurship was the creation of the Norwegian firm Urd Information Technologies.[195] It was definitely a case of an insider from the Chill community – in many ways one of the most ardent supporters of the Chill cause – entering the market through a new venture.

Urd was set up to commercialise products developed within the Nordic Chill compiler project, which the Nordic telecommunication administrations had started in 1977. This activity expanded substantially in the early 1980s, and was related to a larger development system called Chipsy (the Chill Integrated Programming System), a full software development environment. The project also included the British administration, and continued up until the mid-1980s. Eventually, the product would be marketed under the name Chipsy and comprised a set of tools in addition to the compiler first developed, some developed within the realms of Urd, some originating from the continuation of the Nordic compiler project in the early 1980s.[196]

---

[195] The following is based on extensive interviews with Kristen Rekdal, but also the material available in the NTR archives, which contains regular reports on the fortunes on Urd's sales of their Chill technologies, as the administration was one of the owners of the product due to their sponsorship of the Nordic Chill Compiler project. See in particular the two following boxes "L 0135, Samarbeid" and "L 0136, Samarbeid", series "Da, 1961 – 1996", NTR. Furthermore, additional information about the relationship between Runit and Urd was obtained from Svein Hallsteinsen, interview with the author, 21 January 2009, Trondheim, Norway.

[196] For an overview of the Nordic compiler projects, see Rekdal, "The Nordic CHILL Project".

Three exits predated the establishment of Urd. At the micro level, the start-up coincided with Rekdal's exit from the CCITT, as he left his role as chairman of the working group that had been responsible for Chill development in 1984. Furthermore, it coincided with Rekdal's departure from his position at Runit and technical research. At an organisational level, a prerequisite for the creation of Urd was the unwillingness of the Nordic administrations to continue their coordination of the Chill activities, and their unwillingness to market the Chill tools they had developed with Runit. At the macro level, the firm's creation overlapped with the coming of what the historian Lars Thue has described as "the neo-liberal order", a broad transformation of Norwegian politics towards market orientation and the downsizing of the role of the state. This involved the abandonment of the social democratic consensus that had prevailed since the post-war period.[197] The Labour Party left government in 1981, to be replaced by the first majority Conservative government since 1928. More directly, this initiated a change in industrial policy and an increased focus on how applied technical research could be commercialised through spin-offs and firm creation.[198]

In the following, I will substantiate my description of each of these three exits on the political, organisational and individual level. I will also discuss how each exit created an opportunity to commercialise the Chill knowledge that existed at Runit and introduce Chipsy to the marketplace. Let me start at the bottom, at the level immediately experienced by the people involved in the Chill projects.

Kristen Rekdal's exit from Runit came after he had realised that he would not be able to develop Chipsy further at Runit.[199] Runit had "severe problems with funding" and it was believed that the only possibility to develop Chipsy into a more mature product was to commercialise it and spin it off into a separate entity.[200] The increasing maintenance responsibility for the early Chipsy users was in conflict with the research objectives of Runit. By exiting Runit, Rekdal also left the possibility of an academic career as well as the possibility to continue a more theoretical interest in programming

---

[197] The conception of the neo-liberal order has pervaded Thue's work for a number of years, largely in Norwegian. It has also been substantiated in Lars Thue, "Norway: a resource-based and democratic capitalism", in *Creating Nordic capitalism: the business history of a competetive periphery* (Basingstoke: Palgrave Macmillan, 2008).

[198] This is extensively treated in the Norwegian historiography. For an introduction in English, see ———, "Norway: a resource-based and democratic capitalism", in *Creating Nordic Capitalism: The business history of a competetive periphery*, ed. Susanna Fellman, et al. (Basingstoke: Palgrave MacMillan, 2008).

[199] Kristen Rekdal, interview with author, 11 November 2008, Oslo, Norway.

[200] "CHIPSY information meeting", Oslo 15 January 1985, box "L 0135, Samarbeid", series "Da, 1961 – 1996", NTR.

languages and software development, since just before the formation of Urd, Rekdal was appointed as professor at the Norwegian Institute of Technology in Trondheim, a position he never took up.[201]

Rekdal's exit from Runit coincided with the end of his stint as a chairman of the CCITT group that had refined the 1980 version of Chill. By this time, he had also closely watched how the group was unable to fulfil an initial concern towards creating a standardised Chill environment and, more modestly, a standard compiler. He had first-hand experience of the limits of cooperation within the CCITT. He also possessed an almost complete overview of how Chill was used and by whom, since this was reported regularly in the *Chill Bulletin*, of which he was the editor until 1984. If the market for Chill-based tools existed, Rekdal would be the man to know.

To be able to effectuate on this knowledge, ending his liaison with the telecommunication administration, which he represented in the CCITT, while being able to maintain his close relationship with the same organisations, was important. The telecommunication administration had put substantial weight behind the development at Runit. In total, through financing from the Nordic administrations and various industry contracts, the investment totalled 25 million Norwegian kroner between 1975 and 1985.[202] The product, Chipsy, was developed and marketed by Runit, and already in 1979 had been sold to the Norwegian ITT subsidiary STK. In 1981, the sales of Chipsy escalated, with two major orders, one from STK and one from the Swiss company Hasler.[203] In 1982 and 1983 development on Chipsy was substantially influenced by these contracts, and 1984 saw more new customers, in this case the No. 10 Research Institute of the Ministry of Post

[201] "Statsråd ble holdt på Oslo slott 28. oktober 1983", Aftenposten, 29 October 1983.

[202] Stipulations made from Svein Hallsteinsen, "Overview of projects and contracts at RUNIT dealing with CHILL and CHIPSY", 20 November 1985, box "L 0135, Samarbeid", series "Da, 1961 – 1996", NTR. Costs related to the participation in the CCITT are not included and all investments are given in current prices. In prices for 2010 (adjusted with CPI), this would amount to 52 million NOK or 8,605,032 in 2010 US dollars.

[203] In total, these two sales were worth two million Norwegian kroners, and triggered maintenance and development contracts worth 7.7 million Norwegian kroners, in a period from 1981 to 1985. All numbers taken from Svein Hallsteinsen, "Overview of projects and contracts at RUNIT dealing with CHILL and CHIPSY", 20/11 1985, box "L 0135, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR.

and Telecommunications of China, and smaller licenses to the Korea Advanced Institute of Science and Technology.[204]

These activities created a mismatch between the institutional setting and the project. It was stretched between the interests of the owners (the telecommunication administrations) the research institute, Runit, and the commercial activities. This caused concern. In a report from the steering committee in February 1982, this was expressed in the following way:

> The activities inside and outside Runit are now so complex that it is difficult to distinguish between the interests of NT-P [The Nordic coordination group on programming languages, basically the steering committee of the Nordic Chill project], Runit, The Norwegian industry and others. Runit's main activities are neither basic research nor pure industrial projects, but rather applied research and development. Runit must raise money in order to survive, which means that Runit's interest goes where the money is.[205]

The economic squeeze at Runit was well known to the participants in the Nordic Chill project. The steering committee would often report on the economic problems of various sub-projects, which were caused by too ambitious technical goals and contracts with industry partners like STK and Hasler that pushed the project in a direction not necessarily in line with the wishes of the administrations. Generally, Runit lacked both resources and expertise in dealing with commercially applied research and development, which the development of Chipsy resembled more and more, as the increasing amount of maintenance that was involved in the various Chipsy installations would seriously stretch the resources available.

The organisational stretch escalated during the period up until 1984, when the administrations felt that the Chipsy project had run its course and considered cutting the project altogether. In a report from the owners' coordination group, it was argued that:

---

[204] "Rapport no 7 to NT from NT-P for the period 84-03-01—84-09-01", 31 August 1984, box "L 0136, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR.

[205] Søren Werner, "Report from the 10th Meeting on NT-Programming Languages, British Telecom, London, February 9-10, 1982", 10 May 1982, box "NT-Programspråk 1982", KRC.

> CHIPSY has served its purpose. It has been giving experience with CHILL matters to the Nordic countries in their CHILL activities in CCITT. It has supported the common Nordic financed chairman of SWP XI/3-2. Contributions from Nordic delegates in CCITT have also been supported. Furthermore RUNIT as a scientific organization has seen that the administrative tasks of CHIPSY including marketing and selling have become too man consuming in order to continue the way CHIPSY deserves. Therefore CHIPSY could be stopped now.[206]

Despite these objections, there was also considerable interest in continuing the project. Three arguments were used in support. Firstly, Chipsy had attracted far more interest from industry than had been foreseen. The feeling was that while Runit was not able to market and sell Chipsy professionally, it could be "sold to a greater extent if it were taken care of by a professional company". [207] Secondly, there were some indications that the telecommunication administrations in Norway and Finland were planning to use Chipsy for in-house software development. Thirdly, contracts with industrial partners like STK and Hasler obliged Runit to maintain Chipsy for a number of years. The steering committee of the Nordic Chill project concluded: "Chipsy should be kept alive – but handled in an easier way than is the case today."[208] That easier way was to license the rights to Chipsy to an existing firm willing to take it on, or by supporting the creation of a new firm dedicated to marketing, selling and developing Chipsy. Consequently, the committee started searching for an agent. At this point in time, Urd was already in planning.[209]

Even though a tender to "take over" Chipsy was circulated to a number of prospective and existing companies, it was no surprise that the Nordic administrations opted for Urd in the end, since Chipsy basically was

[206] In an appendix to "Rapport no 7 to NT from NT-P for the period 84-03-01—84-09-01", 31 August 1984, NT-213 and NT-P(84)139, box "L 0136, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR.
[207] Ibid.
[208] Ibid.
[209] Urd was discussed at a board meeting at Sintef on 11 April 1984. The business plan that was presented was general, but Chipsy was already mentioned as a point of departure for further product developments in the prospective firm. Ola Nordal, who has written the history of both the Norwegian University of Science and Technology (NTNU) and the computer history of Sintef, lent me a copy of the business plan.

a product of Runit and Urd was a Runit spin-off.[210] By 1985, the newly formed company Urd obtained the rights to sell and further develop Chipsy.[211] The agreement meant that Chipsy remained the property of the administrations involved, and that all sales of Chipsy made by Urd would yield royalties for these owners. However, the administrations could choose to reinvest their net income in further Chipsy developments and would be eligible to use Chipsy free of charge.

The opinion that Runit and the Nordic cooperation was the wrong environment for further developments of Chipsy was not something felt within the administrations. It resonated with a shift in general industrial policy.

In November 1983, the Norwegian minister of labour and local governance, Arne Rettedal, visited the biannual industry conference in Trondheim. A notable conservative politician and former mayor of the oil capital of Norway, Stavanger, Rettedal criticised the dominant attitude in industry and academia in Trondheim for its general lack of risk-taking and market-orientation – something contrary to what he believed was the key to the rise of Stavanger as the dominant city in the rapidly expanding Norwegian offshore oil industry.[212] To remedy this, Rettedal proposed increasing funds for applied technical research and initiatives to increase the efficiency of research and development. In total, the conservative minister claimed that the knowledge institutions of Trondheim had to serve commercial interests in a more effective fashion than before.

At the same conference, Johannes Moe, head of the Foundation for Industrial and Technical Research (Sintef), rhetorically asked whether the organisations in Trondheim engaged in industrial and technical research contributed enough in terms of innovations.[213] Moe claimed they did not, and that Sintef could be an important mediator in a more innovative future.

---

[210] On the tender, see ”CHIPSY information meeting”, 15 January 1985, Oslo, box “L 0135, Samarbeid”, series “Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996”, NTR; Jens R. Rasmussen ”CHIPSY Marketing, Sales and Distribution Companies in Denmark”, 17 January 1985, box “L 0136, Samarbeid”, series “Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996”, NTR. Here, a Danish alternative to Urd is mentioned, Danish Telecom International a/s. Apparently, their offer was not as favourable as that from Urd.

[211] “Agreement between URD Information technology A/S, British telecommunications PLC and Norwegian telecommunications Administration: CHIPSY sale and development”, 13 August 1985, box “L 0135, Samarbeid”, series “Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996”, NTR.

[212] “Rettedalkritikk mot industri og forskning”, Aftenposten, 2 November 1983.

[213] “Johannes Moe under Industridagene: - Samlet kan vi få fremgang”, Aftenposten, 2 November 1983.

Trondheim was a prime target for such criticism, as Norway's largest organisations engaged in technical and industrial education and research were located in the city.[214] As one of many consequences of these critical assessments, a fund for seed money (A/S Etablering og virksomhetsutvikling, Asev for short) was set up in January 1984, only months after Rettedal's visit and Moe's speech.[215] Asev was partly funded by Moe's Sintef, partly from government money and partly by five private banks. Asev's mandate was to help fund and support small spin-offs from Trondheim's academic organisations, and rectify the shortcomings pointed out by Rettedal and Moe.

Only months after Asev's start, Kristen Rekdal was planning to break away from Runit and Sintef. With help from Asev, the creation of Urd was a reality during summer of 1984. The Rettedal and Moe statements in 1983 and the creation of Asev in 1984 illustrate a general shift in research and industrial policy in the early 1980. This involved a policy in which research was to prove its worth through innovations and new firm creation. The idea of creating university spin-offs and subsequently provide knowledge diffusion through the mobility of academic employees was novel and carried out with great optimism.[216] It is worth noting that this belief in spin-offs was not necessarily compatible with the crux of a neo-liberal belief in the effectiveness of markets. One of the prerequisites of this early period of university spin-offs was its reliance on personnel mobility in place of the market exchange of ideas or patents, the epitome of what many an observer has equated with the neo-liberal order.[217] Indeed, the policy of providing institutional and governmental support for spinning off firms from academic institutions could just as well be understood as a stop-gap for market failure.

Just a year after Asev was founded, the new head of the Norwegian Research Council (Norsk Teknisk-Naturvitenskapelig Forskningsråd,

---

[214] Two newly published histories of the main organisations in Trondheim are Thomas Brandt and Ola Nordal, *Turbulens og tankekraft: historien om NTNU* (Oslo: Pax, 2010); Nordal, *Verktøy og vitenskap: datahistorien ved NTNU*.

[215] "Har vist at det går an: Flere nye småbedrifter er etablert", Aftenposten, 12 November 1984.

[216] University spin-offs and their role in knowledge diffusion is a field of scholarly research in its own. See Scott Andrew Shane, *Academic entrepreneurship : university spinoffs and wealth creation*, New horizons in entrepreneurship (Cheltenham, UK ; Northampton, MA: E. Elgar, 2004).

[217] For an explicit discussion about changes in policy towards science in this period, see Philip Mirowski and Esther-Mirjam Sent, *Science bought and sold : essays in the economics of science* (Chicago: University of Chicago Press, 2002). Here, the "globalized privatization regime" is the catch-all phrase, used in a similar sense to the neo-liberal regime above.

NTNF), Inge Johansen, argued that the last few years had entailed an important change to Norwegian research:

> We have noted a change of attitude among our academics during the last few years. They are less modest when it comes to founding their own firms. Many have realized that it is not such a risky undertaking to start on your own. It is great potential to diffuse research results through young people who wish to fund new ventures. Venture capital firms seem to flourish in different regions around the country. These are important instruments to help such entrepreneurs [...].[218]

Funding new firms was, from the viewpoint of the research council, a way to diffuse research results. In that way, the new wave of young people who wished to start on their own was definitively a positive change in terms of fulfilling Rettedal's goal of greater risk-taking and market-orientation. Whether this was down to a change in attitudes or a change in policy instruments and the availability of capital is not so important. What was important was that the change was observable in young people who were eager to start on their own, according to Johansen. One of the examples was Urd, which was trying to move knowledge from the rather closed realms of the Chill community to the market.

## Entering the market

By the summer of 1984, Urd was about to enter the market with their developed version of the Nordic Chill compiler, now a comprehensive programming environment named Chipsy. At the same time, Urd entered the market for financing. Urd was able to secure financing from the seed capital fund Asev. Urd also obtained further financing from Sintef, the owner of the research establishment Runit, and so-called start-up loans from the Christiania Bank og Kreditkasse. The company raised a modest sum initially, but had made provision for future capital needs in the region of 20 million Norwegian kroners.[219] Both rounds of financing drew on existing networks and seed capital rather than the venture capital market per se. In 1984, Urd presented itself in the following manner:

[218] Inge Johansen, quoted in "Adm. direktør Inge Johansen i NTNF: Unge våger nyetablering", Aftenposten, 20 March 1985. My translation.
[219] In 2010 prices (adjusted with CPI), this would amount to 43,960,000 NOK or 7,271,764 in 2010 US dollars.

> Urd Information Technology A.S has the primary objective of doing marketing, sales, production, distribution and customer support of advanced software products and doing consultancy work and development projects in that context. Urd operates in the international marketplace and will primarily focus on the telecommunications sector. The initial product line comprises Chipsy – the Chill Integrated Programming System. Chipsy is a software support environment for Chill – the CCITT High Level Language.[220]

The ambitions were high. The company was to be international from the outset, and it was to base its products on knowledge developed at the same international level. Furthermore, its expertise was within an advanced area of software products that involved huge companies and organisations. The product, Chipsy, had been developed over a number of years and evolved into an advanced and comprehensive programming system or software development environment. After developing Chipsy within the Nordic context from the late 1970s, the product had grown considerably in scope since its inception as a compiler, and was now described as a "software support environment". This meant that a host of different tools was now included in the Chipsy product, like various debugging tools and a run-time system for various target computers.[221] An illustration of the Chipsy system as of 1993 is reproduced below:

[220] "URD Information Technology- A new Software Company to serve you", box "L 0136, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR
[221] For an overview, see Kristen Rekdal, "Catalogue of CHIPSY Products and services", 28 September 1985, box "L 0135, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR

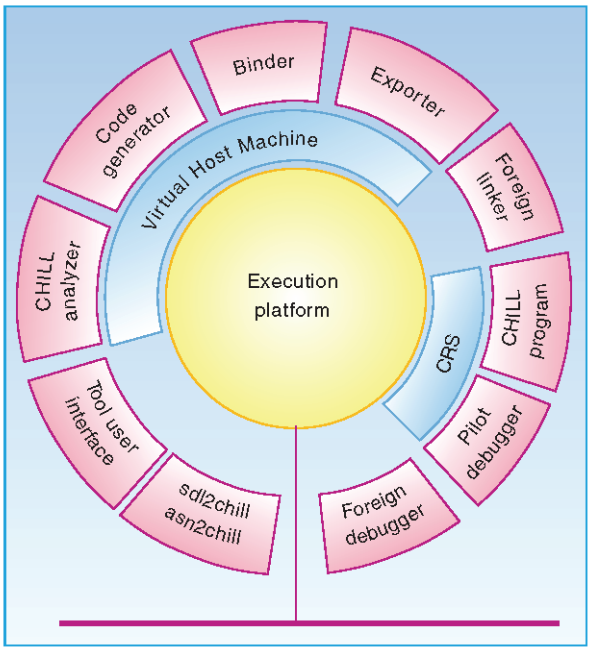


**Figure 7.1 Chipsy as of 1993**

Chipsy had been developed into a comprehensive system that included advanced development technologies. Some of these were unique features, like the concurrent run-time system and a linker system for separately compiled modules. The debugger, called chillscope, was also tied in with the particularities of the programming language, in particular the concurrency and modular concepts.[222]

The move from what was basically just a compiler towards a more integrated support environment mirrored the general shifts in the software engineering field at the time. One indication of this was the aforementioned keynote speech made by Remi Bourgonjon at the 1983 Chill Conference, where he pointed out so-called programming environments as one of four major technical areas where progress was expected.[223] As such, Urd and Chipsy were part of a technical trend. With regard to its commercial possibilities, setting up an independent firm like Urd was rooted in the positive belief and high hopes that the product, Chipsy, would sell in an emerging market.

---

[222] See Svein Hallsteinsen, "Source level debuggers: Experience from the design and implementation of chillscope", in *Advanced Programming Environments*, ed. Reidar Conradi, Tor M. Didriksen, and Dag H. Wanvik (Trondheim: Springer, 1986).

[223] Remi Bourgonjon, ”Programming languages, environments and Chill”, Keynote address at the second Chill Conference, Chicago, March 1983.

Why did Rekdal believe in an emerging market for Chipsy? To answer this, one has to review the signals and indications that led Rekdal and the investors to believe in a market for Chipsy and step back a few years. During March 1982, Rekdal had been on one of his many trips to a CCITT meeting, this time in Melbourne, Australia. On his way, he visited the telecommunication administration of Singapore, some subsidiaries of Norwegian companies abroad and on his way home, he stepped off the plane in Santa Clara, California, to visit Intel, the leading microprocessor manufacturer. Two main impressions stuck out in his report from the trip. Firstly, when visiting the Singaporean telecommunication administration and Jeng Yuan Sheng, Rekdal once again got the impression that Chill was a tool of importance for telecommunication administrations:

> Singapore will take delivery of a new digital SPC exchange in September this year and Singapore had managed to require in the contract that 30 – 40 % of the software should be written in Chill. They had also tried to convince Hitachi, which has already installed a SPC system, to reprogram parts of it in Chill. Hitachi had refused. Even though Singapore is a small country such initiatives are worth noting. Hopefully they are the start of a trend.[224]

As noted in chapter six, the Singaporean administration was hardly at the start of a trend: it was more like they were at the dying end of another one. However, the impression was still a valid one, as administrations looked towards international standards when ordering new digital equipment in the early 1980s. More so, their ability to specify what they wanted from their procurements was considerably more advanced than before, in many ways thanks to international coordination and cooperation. That more and more manufacturers would be interested in a short cut towards Chill-compliant coding was likely. A product like Chipsy would, therefore, be of interest to more and more manufacturers.

The second impression was one not from the telecommunications market, but one from the computer industry. In Santa Clara, Rekdal visited one of the leading suppliers of microprocessors, semiconductors and

[224] Kristen Rekdal, "Travel report from 1. Telecoms, Singapore, 2. CCITT WP VII/3, Melbourne, 3. Nord Computers, Melbourne, 4. Intel Corp, Santa Clara", Runit notat, 13 April 1982, box "NT-P 1982", KRC.

computer electronics, Intel.[225] Here, the idea of independent software suppliers and tool-makers had a strong supporter. Intel represented a rather different type of company, unlike the large integrated firms that dominated telecommunications, as it encouraged the involvement of independent software vendors to supply applications and tools for their product. Unlike most computer manufacturers at the time, Intel did not only encourage vendors of application software, but "fundamental software as well", which implied software development tools like compilers, as made clear by Rekdal in his report from the visit.[226] This meant that Intel, a major supplier of microprocessor equipment to other computer manufacturers, was open to the commercial exchange of compilers and development tools from outside firms, although the firm also supplied their own range of tools.

The impression Rekdal got from his discussions with the company was that "Intel will not try to compete with [the] independent software vendors, but rather have a strong cooperation."[227] To Rekdal, this was an encouragement, and furthermore, he received solid promises of actual backing from the company. Rekdal noted: "Intel is willing to lend us hardware and software needed to complete the bootstrapping of Chipsy onto iAPX86."[228] This strong support for independent software vendors of programming tools from one of the leading American firms was another indication of a future market to Rekdal, as he envisioned telecommunications as something that would slowly move in the direction of the computer industry structure. To support the idea of Urd as an independent tool-making firm in such a future market, the fact that throughout the 1970s, software development tool-makers had been steadily able to find independent positions in the American computer market, as

[225] In the early 1980s Intel was not a dominant supplier of semiconductors, but rather a pacesetter and the original microprocessor producer. For an overview of the history of the semiconductor industry, see Ernest Braun and Stuart Macdonald, *Revolution in miniature : the history and impact of semiconductor electronics*, 2nd ed. (Cambridge Cambridgeshire ; New York: Cambridge University Press, 1982); Richard N. Langlois and W. Edward Steinmueller, "The Evolution of Competetive Advantage in the Worldwide Semiconductor iNDUSTRY, 1947 - 1996", in *Richard R. Nelson*, ed. David C. Mowery (Cambridge: Cambridge University Press, 1999). On the history of Intel, see Ceruzzi, *A history of modern computing*.

[226] Kristen Rekdal, "Travel report from 1. Telecoms, Singapore, 2. CCITT WP VII/3, Melbourne, 3. Nord Computers, Melbourne, 4. Intel Corp, Santa Clara", Runit notat, 13. April 1982, box "NT-P 1982", KRC.

[227] Ibid.

[228] Ibid.

illustrated by the fact that some of the very first traded software products were products aimed at software developers, was an encouragement[229]

Rekdal encountered what he thought to be the two most important elements of a market for Chipsy on this trip: administrations pushing for Chill and hardware manufacturers interested in independent software vendors supplying "fundamental software". The interest from small telecommunication manufacturers – like Hasler and STK – had already created an impression of a specific place in this market for Chipsy. Both STK and Hasler used Chill and Chipsy to programme systems that would be part of non-public switching, in the case of Hasler a telex machine, in the case of STK, a PABX and a military digital telephone switch, equipment sold in expanding and non-regulated markets. Furthermore, a contract amounting to somewhere in between two and three million Norwegian kroners sold Chipsy as part of a deal done by the Norwegian computing manufacturer Norsk Data to a research establishment in China made the involved parties optimistic over an international future.[230]

The belief in entering the market with Chipsy was built on two pillars. Firstly, experience from selling Chipsy to a number of early adapters, like the Norwegian ITT subsidiary STK and the Swiss manufacturer Hasler was understood as an indication of the viability of selling the Chipsy system as a product. Secondly, the impression of a future in which Chill was something the administrations would insist on was still realistic until the mid-1980s. Furthermore, the established structure of the computer industry, where independent tool-vendors had an important role to play, was feasible in telecommunications.

The decision-making context in which Kristen Rekdal acted was a changing environment, a context that moved on the continuum from uncertainty to risk, but not only in one direction. In 1982, the idea of Chill as an administration-enforced standard was probable, and there were several indications that the market for Chill tools was quite viable. Many manufacturers without prior Chill competence could be forced to programme their new equipment in Chill, and buying a product that could ease this would certainly be a viable strategy. This created an opportunity to market Chill to two types of organisations: administrations with heightened ambitions and manufacturing firms challenging for new market openings.

---

[229] Applied Data Research's Autoflow is generally considered one of the first software *products*. See Campbell-Kelly, *From airline reservations to Sonic the Hedgehog : a history of the software industry*, 89-119.

[230] Kristen Rekdal, interview with author, 28 November 2007, Oslo, Norway. In 2010 prices (adjusted with CPI), this would amount to something between 4,675,136 and 7,012,704 NOK or something between 773,376 and 1,160,064 in 2010 US dollars.

However, as the firm was set into motion, these probabilities changed dramatically, and the possibility to sell a tool like Chipsy to administrations became either unrealistic or highly uncertain. Things did not work out.

## Hard times

Sales were hard to find after Chipsy was spun off from Runit and into Urd. The international marketplace and the telecommunication sector were no easy place to be in and Chipsy was not easy to sell. In other words, Urd had a rough start. Just eight months after the company started selling Chipsy, the future looked grim. Kristen Rekdal reported on the problems to the representatives of the Nordic administrations, which still met regularly in the group of Chipsy stakeholders:

> After having actively marketed CHIPSY for 8 months, there is no sign that the market will take off soon. Although considerable interest has been shown in the product and more than 30 potential customers identified, no new customers for full binary or source licenses have been signed up, so far. There is a great reluctance in the marketplace to invest in new programming languages or tools based on technical merits only.[231]

Obviously, the plans for Chipsy and Urd had not panned out. What had changed since the inception of Urd and the industrial pressure for "professional marketing and support", which had made it impossible to continue supporting Chipsy within Runit and the supporting administrations?

While the sales of Chipsy to STK and Hasler created a situation where the Nordic owners wanted to get rid of it due to the administrative workload, such sales were apparently hard to come by after the creation of Urd. Chipsy sales during 1986 give us some indications of a lacklustre performance: by 1986, Urd supported 32 Chipsy users, of which 25 had Chipsy supplied from Urd. The others were delivered by Runit before 1985. However, none of the 25 deliveries made by Urd was to a substantial industrial company that would be eligible for a so-called full binary license, which would be the most lucrative customers, but to users that were eligible for discount licenses. These were either strategic placements of Chipsy at educational institutions, deliveries to the Chipsy owners (the five administrations that had financed the project were initially eligible for free licenses), or deliveries to existing users like STK and Hasler. During 1986 Urd held talks on major development projects with the German firm Nixdorft AG, the Swedish company Telenova, the British GEC and the Indian C-DOT research

[231] Kristen Rekdal to NT-P (Owner's reps), "Status report on CHIPSY Marketing and Sales", 31 October 1986, box "L 0136, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR.

establishment, without any luck.[232] What caused these problems? Chipsy was intimately tied in with Chill, which was both its strongest selling point and its biggest problem. By the mid-1980s, the general market for programming tools for real-time and telecommunication systems was expanding. Tools for the competing programming language Ada was popping up regularly.[233] However, there was no real impetus for this market to be directed towards Chill, an obvious problem for Urd. As such, the market was there, but not in the form envisioned by Rekdal a few years back. Rekdal was, however, optimistic that such a move would happen, eventually. "Given the general increase of CHILL activity world wide, the CHILL market is bound to mature, but the timing is difficult to forecast," he noted.[234] The interest in Chill came mainly from large industrial firms, such as Siemens, the ITT and Philips, and rested on the belief that it would become a mandatory requirement from the administrations. These large firms did not depend on market exchange to obtain Chill tools. They developed the necessary tools themselves, much like the origin of the capital tool industry in the USA during the second industrial revolution.[235] Rekdal marketed Chipsy in a market that did not exist at the time. As evident from the encounter with the telecommunication administration of Singapore, his hope was that this market would be created with the aid of the telecommunication administrations, either in terms of mandatory requirements for future procurements or through their use of in-house development (and consequently, their need for development tools). Rekdal formulated this in the following way when he addressed the owners of Chill in 1986:

> URD Information Technology is a company too small to develop the CHILL market alone [...] It is necessary to have a much stronger backing from the telecom administrations, in particular the Owners. Otherwise it is not possible to generate the income necessary to keep CHIPSY alive while waiting for the market to expand.[236]

The ill-fated attempt to sell Chipsy in the market was intimately bound up

---

[232] Kristen Rekdal, "Status report on CHIPSY Marketing and Sales", 31 October 1986, box "L 0136, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR.
[233] Sammet, "Why Ada is not just another programming language".
[234] Kristen Rekdal, "Status report on CHIPSY Marketing and Sales", 31 October 1986, box "L 0136, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR.
[235] Rosenberg, "Technological Change in the Machine Tool Industry, 1840 - 1910".
[236] Kristen Rekdal, "Status report on CHIPSY Marketing and Sales", 31 October 1986, box "L 0136, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR.

with the ambiguous policy of the administrations, and in particular the five that owned Chipsy: the Nordic administrations plus the British one. Rekdal's call on the administrations to take a greater interest in Chill and Chipsy brought about nothing more than uncommitted formulations like the following, made by the Finish representative Seppo Ylä-Pietila: "Concerning backing CHIPSY, all we can do, and have done is to recommend our organisations to use CHILL."[237] However, Pietila also made it clear that Rekdal's summary of the market situation was received "with great astonishment".[238] The participants in the Nordic Chill project had also believed in the prospective market, but could do little to create it themselves.

To Urd, neither uncommitted recommendations nor astonishment were enough. When the Norwegian telecommunication administration was considering a tender for their pilot ISDN project in 1986, Urd was hoping for a change. In a letter to the powerful head of the Norwegian Telecommunication Research Establishment, Nic Knudtzon, Kristen Rekdal argued that the ISDN software had to be based on Chill, and that this could be an important outlet for the Norwegian industry.[239] However, no such preference was articulated by the administration in their invitation to participate in their ISDN pilot. The lack of a positive response led the Urd chairman to address the ministry of industry, Finn Kristensen:

> The ISDN project represents a unique opportunity to build an internationally oriented software activity based on existing [Chill] competence. One condition would be that the government services such as Televerket [the administration], through high ambitions within the field of software and an extensive use of international standards, would support such industry activities.[240]

However, no intervention from the minister was forthcoming and as discussed in chapter six, the Norwegian administration never took a strong stance on the issue of programming languages. Urd was left with a market that did not move in the direction it had planned for.

In 1990, Rekdal would concede: "The market has not at all fulfilled expectations. License sales have been at best sluggish and are likely to

---

[237] Seppo Yla-Pietila – Kristen Rekdal, Norwich, 27th November 1986, box "L 0136, Samarbeid", series "Da, 1961 – 1996", NTR.
[238] Ibid.
[239] Kristen Rekdal – Nic. Knudtzon, 6 November 1986, box "L 0136, Samarbeid", series "Da, 1961 – 1996", NTR.
[240] Erik Amble - Finn Kristensen, 21 November 1986, box "L 0136, Samarbeid", series "Da, 1961 – 1996", NTR.

remain so [...]"[241] Consequently, the license agreement with the Nordic administrations was cancelled, and Urd (now named Kvatro) would go on to focus on different products and services. Pietila, summarised the fate of Chipsy in the following way:

> I also have a feeling of disappointment at CHILL and CHIPSY not having had more general support in the world of telecommunications. May be the main reason is that manufacturers (many of them) already had their development tools chosen when CHILL came along. The lack of CHILL training in Universities in general and lack of widely spread CHILL tools have also had their impact on decisions made concerning the support of CHILL. De facto standard languages such as C and Pascal seem to have overrun [the] actual standard language.[242]

Pietila's summary seems apt: Chipsy was intimately bound up with Chill. In the mid-1980s, it experienced strong competition from C and Ada. Regardless of which programming language triumphed, the viable market for Chipsy in the second half of the 1980s was limited: the large manufactures developed the tools they needed in-house, and the possibilities of them opening up were slim. Furthermore, the administrations never got around to doing much programming on their own, which made them unlikely customers. The few customers were the small and medium-sized companies, like STK and Hasler, companies that had already been recruited before the firm's creation.

Could the reason be that Chipsy was not a very good product? Whether the compilers developed by firms like Philips, Siemens and the ITT, or the one distributed and sold by Imperial Software, were "objectively" better than the one developed by Runit and Urd is difficult to judge in retrospect. Among other things, it would necessitate the availability of a rather esoteric set of hardware and the availability of a reasonably complex part of code to be able to judge the compiler's efficiency (in terms of compile time and amount of compiled code) and the ability to search out erroneous code. However, I find few reasons to doubt the quality of the software: The Urd compiler was a well-tested compiler at the time of its commercialisation, and was used by many outside Runit, and I think such use would be improbable had the system not worked. Furthermore, while user comments about the state of Chipsy were certainly not all positive all the time, it seems the product was valuable enough to make industrial users pay for it long before it was introduced to the general telecommunication

[241] Kristen Rekdal to NT-P, "Status Report on Chipsy License Sales 1989", 30 May 1990, box "L 0136, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR.
[242] Seppo Ylä-Pietila to Kristen Rekdal, Telefax, 5 November 1990, box "L 0136, Samarbeid", series "Da Sakarkiv ordnet etter arkivnøkkel, 1961 – 1996", NTR.

market. The compiler developed by the Bjørner group and Imperial Software was known to produce effective code through its translation of the source code into machine code.[243]

Despite all the misfortune with Chipsy, Urd continued to hang on, but the company had to continue to look for new opportunities. While maintenance and consulting for Chipsy customers continued throughout the 1980s and into the 1990s, the small growth resulting from these activities was unsatisfactory for both owners and employees.[244] One of the consequences of this was an owner-led intervention in 1988 that reorganised the firm substantially. After the initial seed capital financing, the venture capital fund Origo became one of the owners of Urd. By 1987, Origo reorganised Urd and merged it into another of its companies called Kvatro (Kongsberg Våpenfabrikk avdeling Trondheim). Kvatro had been established as a Trondheim-based outpost of the state-owned armament factory Konsberg Våpenfabrikk (KV) in the mid-1980s, as a means of recruiting programmers directly from the city's technical university. The main task for Kvatro was programming embedded military systems produced by KV and it employed about 50 programmers by 1987, when the state-owned Kongsberg Våpenfabrikk capsized in what has become the epitome of the failed industrial policy that was associated with the interventionist Labour party in the 1970s.[245] After this a number of the armament factory's divisions were reorganised or sold off, including Kvatro. Kvatro was initially sold off to its employees, but later on helped through by the venture capital fund Origo, which also had bought parts of Urd previously.[246] One requirement for Origo's financing of Kvatro was to merge Kvatro and Urd, a merger that was completed in 1988.

Kvatro and Urd held, at least technically, some similarities. Kvatro was programming embedded military computer systems, which held many similarities to telecommunication switching systems. Some of KV's systems were even programmed in Mary, the Runit-created precursor to Chill. The

---

[243] A casual comparison between code generated by the Chill compiler and a standard C compiler is presented in Peter J. Smith, "Experiences in achiving a full implementation of Chill", in *Fourth CHILL Conference* (Munich: Siemens AG, 1996). Here it is argued that the Chill programs were 25 per cent faster than a similar program written in C and compiled by a standard C compiler. Although the use of benchmarks like this tests the combination of the programming language *and* the compiler, in this case it produces a favourable result in both respects.

[244] Parts of the following section are based on Kristen Rekdal, interview with author, 28 November 2007, Oslo, Norway.

[245] The capsizing of Kongsberg Våpenfabrikk is one of the subjects of a coming business history of the firm written by Knut Øyangen.

[246] Kristen Rekdal, interview with author, 28 November 2007, Oslo, Norway.

result was a merger between Kvatro and Urd, with the former name taking precedence.

However, the technical similarities were not enough to secure renewed success and growth. The market for Kvatro's original military systems programmers was shrinking rapidly, in particular because of the severe problems with KV. The business of the telecommunication division depended on the long-term contracts won by Urd and even Runit before the merger. However, none of these contracts led to growth.

The envisioned benefits of merging the two lines of business did not materialise, as the technical similarities were not matched by market convergence. The consequence was downsizing, in particular in the original Kvatro part of the company, which ended up with no more than about 30 employees when it finally reached the bottom and started growing again from 1990.

Despite all the misfortune with Chipsy, Kvatro continued telecommunication operations, some related to Chill, and even some further developments to Chipsy. In particular, the developments focused on targeting new host platforms, meaning the systems on which Chipsy was able to run. In 1990, several Unix platforms were able to run Chipsy and by 1992, new debugging technologies and modelling tools were integrated into the Chipsy system.[247] The telecommunication division of Kvatro was spared the downsizing of the late 1980s, as its long-term contracts made it possible to develop new products. In the early 1990s, it fortunes improved slightly. In 1992 Kvatro signed a substantial contract with the NTT, a contract that led to Kvatro being awarded a prize for securing exports and was hailed as an example for an internationally oriented industry in the company's host town, Trondheim.[248]

The development contract was related to an advanced debugging system for telecommunication systems called Pilot, a product that was closely related to Chipsy but was still substantially novel and advanced compared with what the Japanese administration or any of its industrial partners were able to develop on their own. Later on, contracts for Pilot were signed with the Korean research institute ETRI and with the companies Samsung, LG and Hanwha. This product would later on be the main spur for two buy-outs of the telecommunication part of Kvatro, the first by the Finish consultancy firm TietoEnator in 1997, and in 1999 by the Swedish company Telelogic. At the height of the telecommunication bubble in 2001, the

[247] Rekdal, "CHILL - The International Standard Language for Telecommunications Programming".
[248] Kristen Rekdal, interview with author, 28 November 2007, Oslo, Norway.

Norwegian subsidiary was first downsized and ultimately closed, and much of the telecommunication activities finally disappeared.[249]

The alert reader will immediately see the irony in this: Telelogic was initially the company created by the Swedish telecommunication administration to capitalise on the programming language Ada (created for the American military) and on the SDL definition language that were developed in the CCITT in parallel with Chill. Telelogic's development of Ada tools was toned down and later on scrapped altogether, and eventually the company was spun off of from its tight relationship with the Swedish telecommunication administration. When Telelogic bought Kvatro Telecom in 1999, this development was finally brought full circle. The diffusion of Chill was once hampered severely by the lack of support from the Swedish industry, and in particular the way Telelogic preferred to concentrate on Ada. When buying Kvatro, Telelogic was mainly interested in Kvatro's SDL expertise and the debugging system Pilot, which was also what had attracted TietoEnator in the first place.[250]

To complete this, Telelogic was itself the subject of an acquisition in 2008, by IBM.[251] All in all, the venture founded by Kristen Rekdal in 1984 would implicitly end up in IBM by 2008, through a number of market exits and entries, buy-outs and mergers.

## The viability of independent vendors

Large manufacturers like ITT, Siemens and Philips committed themselves to a specific Chill subset very early on, which basically left them with fairly specific implementation tools and incapable of or uninterested in utilising tools made by independent suppliers. However, firms in the midst of the upheavals of the equipment industry, like the British GEC, did turn towards outside vendors for their Chill needs, as exemplified by the involvement of the Danish Chill compiler, which was eventually further developed by Imperial Software for GEC. Nevertheless, what had looked like a real opportunity became unfeasible just a few years later. The chance to sell Chill-based products did exist, but it was an opportunity far smaller than first envisioned. It involved selling Chill tools to small equipment manufacturers, like the Swiss-based Hasler, and to other types of organisations, like Urd's sales to Chinese research establishments. This leads inevitably to question whether the problems of selling such software tools to large telecommunication manufacturers were specifically related to Chill, or to the telecommunication industry itself. None of the other tool vendors got far

[249] "Sier opp 20 av 55", Adresseavisen 17 July 2001.
[250] Kristen Rekdal, interview with author, 28 November 2007, Oslo Norway.
[251] "IBM Completes Acquisition of Telelogic AB," Press release, 3 April 2008, available at http://http://www.telelogic.com/, (retrieved 3 August, 2008).

with Chill-related tools, although Imperial Software continued and developed a range of other software development products. Other companies were, however, able to carve out a niche by delivering tools to telecommunication manufacturers. The aforementioned Telelogic, with its SDL products, was just one example. Urd also followed that route, investing in tools that transformed SDL code into Chill code. A casual analysis would at least indicate that it was not completely impossible to sell tools to the manufacturers, while Chill was a particularly difficult realm.

Ada can again serve as a comparative example. Just as the Chill community and the Chill marked struggled to take off, the DoD-sponsored Ada project never made the huge impact that had been expected in the 1980s. According to one central participant in the Ada project, there was no rapid growth of components and tools for that programming language either:

> However, upon adoption of the common language, I had envisioned a rapid growth of components and tools within a cooperating community, an Ada culture. It did not happen as quickly as I had hoped. 'Repository' is now a universal buzzword, but DoD contracting limits (and the mindset that these have built up) long strangled the vision for cooperation and growth. I mistakenly thought that with the influence of the DoD we could pull it of. The thrust of cooperative development struggled and faded, several times.[252]

This was experienced by an organisation we have already encountered, namely the aforementioned Dansk Datamatik Center (DDC). DDC was the creation of Dines Bjørner, as a sideline to his stint as a professor in computer science at the Danish technical university. It was set up in the late 1970s, and was instrumental in developing the Chill compiler that was taken over by Imperial Software. Its main line of business, however, was related to Ada. In the early 1980s, DDC was heavily involved in applied research on Ada compilers, which were eventually sold through DDC International by the mid-1980s. Compilers were sold to industrial firms like Nokia, Honeywell and NEC. However, by 1990, DDC was closed down. The company failed to attract Danish customers to the Ada tools, and the EEC and government funding were not enough to run the commercial operations of DDC.[253]

However, the sustainability of the independent tool vendor model of business was nevertheless more viable for Ada than for Chill. One of the most successful ones, Rational Machines, ended up in IBM, as did Telelogic, in 2002. Rational started out as a firm specialising in Ada tools in 1981, and

---

[252] Whitaker, "Ada—the project: the DoD high order language working group".

[253] The history of DDC was presented by Dines Bjørner, Christian Gram, Leif Rystrøm and Ole Oest at the third History of Nordic Computing Conference, 18 – 20 October 2010, Stockholm. The proceedings are to be published by Springer in the spring of 2011.

by the late 1980s, it was one of many companies selling such tools that were not involved in hardware developments at the same time. By 2002, IBM bought Rational Software for about 2.1 billion dollars.[254] On its way, it had bought Ivar Jacobson's Objectory AB from Ericsson. Ivar Jacobson had, as described in chapter four, been instrumental in the design of the concurrency features in Chill.[255]

This illustrates the viability of firms specialised in software tools for the telecommunication industry. Chill was, however, seemingly too tied up in the "ancien regime" to be a viable platform for extensive entrepreneurship. The opportunities were limited and the eventualities of the 1980s did not pan out in its favour.

## The long postlude

The closing down of the remains of Urd in 2001 marked the end of the tool-selling opportunity for further Chill diffusion. The final closure of Urd was also the end of all efforts to commercialise Chill-related products. In many ways it did mark the end of the Chill life cycle, beyond the last ITU-approved recommendation of 1999. Throughout the hard times of the 1990s, Chill was still maintained and improved upon in the CCITT. As seen in chapter six, important technical improvements were made to the language as late as 1996, when object-oriented features were added. However, by 1999, CCITT published what was to become the last version of the programming language. As such, the demise of Urd and the final withdrawal of the Chill compiler from the free Gnu compiler collection, both in 2001, mark some sort of closure. By that time, no new developments were done with Chill.

All in all, Chill petered out by the end of the 1990s. It tailed off and diminished into nothing. However, its demise was far slower than what would normally be expected for to a failure. Because of the longevity of many telecommunication systems, the remains of the programming language are still ticking along in old switching systems still operating around the world. The legacy code in these systems has to be maintained like other elements in technological systems, although the maintenance of software is something completely different from the common repair job. More often, repairs to Chill software arise from responses to changes in the extended technical environment of the telecommunication systems, changes that were

[254] "IBM Completes Acquisition of Rational Software," http://www.ibm.com/ (retrieved 8 March 2011).

[255] On Ivar Jacobson's involvement with Rational, see the interview with Jacobson in Biancuzzi and Warden, eds., *Masterminds of Programming* 317-74.

not anticipated by the original software developers.[256] Dealing with legacy code written in ancient programming languages like Cobol and Fortran has been a pervasive part of software development. The Year 2000 debacle, when the software industry franticly updated old software so it could distinguish properly between the years 2000 and 1900, is just one of many recent examples.[257] That similar updates of antiqued telecommunication equipment programmed in Chill, although not related to the year 2000 problems directly, have been common, is not in doubt. As such, a programming language never dies, but rather just peters out. In this case, the fate of Chill seems quite common.

As Chill became legacy code, the idea of so-called concurrent programming languages also faltered. The general technological change of languages that catered for the perceived needs of large communication systems and real-time computing, languages like Modula, Chill and Ada never gained a solid grounding, at least not to become commonplace in general computing.

Ada and Chill had other things in common: their early standardisation and focus on concurrency did not make them the end-all languages of real-time and embedded computing. When the US Department of Defense commissioned the Ada programming language in the late 1970s, the idea was to mandate its use across all the services. Despite this, few programmers used Ada, and it was finally dropped as a mandate by the Department of Defense in 1997.[258]

In contrast, the programming language developed at the AT&T's Bell Labs, C, became a huge success throughout the 1980s and 1990s, also as a systems programming language for embedded and real-time systems. The success came despite the fact that C was, according to its designer, quirky and flawed.[259] According to another well-known computer scientist, Niklaus Wirth, C did not represent much of an improvement at all, as it certainly did not raise the level of abstractions for the programmers.[260] Yet C certainly became a huge success without the huge range of features that Ada and Chill shared. Its feature set and scope were indeed quite small. Its more advanced

---

[256] The role of software maintenance has been investigated by Nathan Ensmenger. See Ensmenger, *The computer boys take over : computers, programmers, and the politics of technical expertise*, 223-27.

[257] On legacy software, see Michael S. Mahoney, "What Makes the History of Software Hard", *IEEE Annals of the History of Computing* 2008.

[258] The mandate was effectively ended with what is known as the Paige memorandum. See Emmett Paige, Jr., "Use of the Ada Programming Language," 29 April 1997, ASD(C3I). The Ada Joint Program Office (AJPO) closed 1 October 1998.

[259] Ritchie, "The Development of the C Language".

[260] Wirth, "A Brief History of Software Engineering".

and object-oriented sibling, C++, became widely adopted.[261] Adoption was, however, not the same as success. C++ It was famously used in L. M. Ericsson's failing second-generation Axe switches, which completely capsized by 1995.[262]

An almost paradoxical extension to the C programming language was Objective C, which was first developed by researchers at ITT's programming technology centre in Stratford, Connecticut in conjunction with the company's use of Chill. Objective C was eventually used as a system programming language for the NeXT Computer, Steve Jobs's project after quitting Apple. This would later form the base of Apple's OS X operating system, and Objective C would be the main programming language for development for Apple's iPhone. In a very indirect manner, a C-based programming language made at a telecommunication manufacturer became a key technology when Apple entered the market for mobile telephones in the late 2000s.[263]

To testify further to the problems of getting the idea of such specialised programming languages to stick, we can turn to one of the most radical and late entrants to the pack, the programming language Erlang.[264]At the 1990 International Switching Symposium (ISS), the new programming language was presented hot off the shelves of the Computer Science Laboratory of L. M. Ericsson, incorporating the latest and greatest strides in concurrent programming language design.[265] After eight years of tentative use within the firm and on one major switching project, L. M. Ericsson put

---

[261] On the history of C++, see Bjarne Stroustrup, "A history of C++: 1979 - 1991", *ACM SIGPLAN Notices* 28, no. 3 (1993); ———, "Evolving a language in and for the real world: C++ 1991-2006", in *Proceedings of the third ACM SIGPLAN conference on History of programming languages* (San Diego, California: ACM Press, 2007).

[262] The AXE-N venture was to be the most expensive industrial project in Sweden after Saab's JAS fighter. The project has often been described as a total failure. See Sven Olof Karlsson and Anders Lugn, *Changing the world : the story of Lars Magnus Ericsson and his successors* (Stockholm: Sellin & partner, 2009).

[263] The development of Objective C and its relationship to ITT are discussed in chapter five. See also the interview with Brian Cox and Tom Love, the language designers, in Biancuzzi and Warden, eds., *Masterminds of Programming* 241-76.

[264] On the history of Erlang, I rely on Däcker, "Concurrent Functional Programming for Telecommunications: A Case Study of Technology Introduction"; Joe Armstrong, "A history of Erlang" (New York, NY, USA, 2007).

[265] Erlang is, apart from being a programming language with a strong focus on concurrency, quite different from the aforementioned languages, as it is a so-called functional programming language. This makes it an odd match for the languages discussed above. However, it is at least one in a long line of programming languages initially made for telecommunication systems, which makes it part of the Chill family.

the lid on the technology and banned it from further use within the company. By that time, L. M. Ericsson had decided on a policy of outsourcing software tools development to the American firm Rational, finally opting for a policy that independent tool vendors like Urd had hoped for 10 years earlier.

Although Erlang would make a strong comeback as an open source programming language that diffused to a large number of firms in the following decade, the late 1990s seem like a natural end point of this story.[266] This is not so only for Chill, but for a host of concurrent programming languages in general. By that time, the role of the programming language had also been downgraded in general, and the role of software methodology and tools was promoted to such a degree that some would argue that the choice of programming language was the least important matter.[267] Furthermore, as the re-use of and open sourcing of important software components became all the more common from the late 1990s, the goals of the CCITT, and the virtues of the technological practitioners, were fulfilled by other means. However, tools and methodologies that were too language specific, like Chipsy, would share the fate of their antiqued languages.

The long postlude of Chill and the general demise of programming languages that were designed in the mid-1970s with concurrency in mind pay testimony to the unruly nature of the direction of technological change. In the mid-1980s, this period was looked upon with awe, as evident in a book by Judy Bishop from 1986. Here, it is argued that

> the period from the mid-1970s to the early 1980s was one of immense change and development in programming language design. The host of Pascal derivates launched during this period [.....] all aspired to the three goals of reliability, understandability and verifiability. The achievement of these goals rested on the resolution of the new language issues of data abstraction and formal specification, but also led to a renewed look at accepted features such as data types, operators, loops, exceptions, input/output and modularity. The culmination of much of this research is embodied in one language which is destined to become widely available – Ada.[268]

In retrospect, this period of "immense change and development" looks more

---

[266] On the surge of interest in Erlang after its separation from Ericsson and the Computer Science Laboratory, see Däcker, "Concurrent Functional Programming for Telecommunications: A Case Study of Technology Introduction", 39-44. More recent indications of this are given in Armstrong, "A history of Erlang".

[267] On this move to repudiate the importance of programming languages, see M. Ben-Ari, *Understanding programming languages* (Chichester ; New York: Wiley, 1996).

[268] Bishop, *Data abstraction in programming languages*, vii.

like a period of liminality, and to some extent, a period where the importance of programming language design waned in general.

## Some conclusions

The three last chapters have surveyed some paradoxical effects of the re-regulation and transformation of the telecommunication industries: Most obviously, the upheavals of the 1980s were followed by a contraction in the telecommunication equipment industry, as established manufacturers merged or took part in strategic alliances, effectively reducing the number of market players. Secondly, the administrations that wished for more control over their own equipment in the 1970, in particularly those in the smaller markets, would defect from the cause just as the ties to the strong manufacturers were severed. Rather then becoming more technically savvy organisations able to develop the switching software themselves, the network operators of the 1990s became all the more dependent on the technological capabilities of their suppliers. Thirdly, while new tools that would ease the use of Chill matured throughout the 1980s, they were bought by smaller manufacturers, like the Swiss Hasler, firms that did not internationalise to any large extent. As such, it seemed the market that start-ups like Urd had hoped for did not materialise.

Urd's rise and stuttering life was due to the same cause: the re-regulation of the telecommunication market signalled a substantial opportunity for commercialisation of Chill-oriented tools, as the early sales of Chipsy indicated. However, the same re-regulation did not open up the large established manufacturers to outside vendors, as they instead contracted and merged with each other. In some ways, Urd was a double casualty of the upheavals of the 1980s: the entrepreneurial route of an independent toolmaker was one very much in line with the impetus behind the re-regulation of the telecommunication industry, but the consequences of this change did not favour its disciples.

The number of entrepreneurial firms emerging from the Chill community or exploiting the opportunities of Chill was limited. During the implementation phase in the late 1970s, about 12 compiler construction projects were started. That two of these would end up as spin-offs from applied research settings is, perhaps, not so impressive. However, the limited number of entrepreneurial firms stemming from the Chill project has partly been dealt with in the previous chapter, which highlighted the rapid internalisation of tools within large manufacturing firms. Of the 12 trial implementations, four would go on be used extensively within firms, and only a couple of the other projects had a similar organisational footing as the two that spun off from independent research institutions.

The selection of entrepreneurial ideas was intimately related to a set of virtues esteemed within the technical community of Chill users. Firstly, both the Imperial Software system and Chipsy were related to actors with a strong

bent towards computer science and a mathematically oriented software development virtue. More so in the case of the Danish compiler per se, as it was based on the work initiated by Dines Bjørner, but the work at Runit was steeped in European computer science just as much. However, the entrepreneurial activities were more influenced by a particular understanding and belief of the strategies towards the division of programming labour between administrations, manufacturers and independent toolmakers. The opportunities inherent in the break up of the "ancien regime" were discovered and acted on in countries with no strong links between a nationally oriented manufacturer and rather weak administrations, such as Denmark and Norway.

This involves a paradoxical conclusion: parts of the Chill community had pursued an approach to software development marked by formalism. In one way, this would ease technology development as it made knowledge explicit. The formalism was, indeed, practical. However, this could also be understood as a hindrance to further entrepreneurship, since the knowledge was easily diffused within the community and no one would be more "in the know" than anyone else. On the other hand, the formalism applied was of such an astute structure that a very strong background in computer science formalism, or membership of the Chill community, was necessary to be able to draw on it. As such, entrepreneurship coming from outside the community never materialised. Those who were able to draw on this were first and foremost employees of the large manufacturing firms. In some ways, these firms were really those that were freed from the "ancien regime" in the 1980s and the entrepreneurial possibilities were not for everyone to grasp.

# 8. Conclusions

In 1974, the ITU decided that the organisation should make a common programming language for all public telephone exchanges. In 1980, after more than six years of work, the ITU published their recommended programming language as an international standard called Chill. In 1990, Chill was the only programming language used by more than one manufacturer of public switching equipment. It was used in successful switching systems like ITT's System 12 and the Siemens EWSD. More than 12,000 programmers had been acquainted with the programming language. By 1999 the recommendation was no longer maintained by the ITU. From a highpoint of somewhat widespread adoption, the language slowly faded out and diminished into very little. An imagined obituary would have concluded that it reached the age of 25, that it had led a troubled life but passed away peacefully. How did this happen?

This thesis has accounted for the emergence, diffusion and demise of Chill by analysing the changing political regimes of telecommunications, the role and ideals of different technical communities and the influence of the strategies of various telecommunication organisations. I have tried to integrate these levels of analysis into an account of the technical diplomacy that went on within and around the Chill project. This has highlighted how and why the programming of telecommunication equipment, in general, was developed in the direction of high-level languages like Chill. As the process has been analysed as technical diplomacy, the participants have been characterised as ambassadors of their professions or smaller technical communities. They also held loyalties to their organisations, be they telecommunication manufacturers, administrations or more independent research establishments. Which loyalties held the highest currency, and at what time they enjoyed the greatest legitimacy, have been the general mode of explanation of the direction of technological change analysed in this thesis. Three periods have been looked into in detail: the design and implementation phase of emergence, the diffusion years and the long postlude of demise. Together, these three phases spanned the 25 years between 1974 and 1999.

The initial phases of the design of Chill started, in its earnest, in the first half of the 1970s, and were intensified from 1974 to 1976. Together with the years of trial implementations, officially lasting from 1977 to 1980, this period makes up the phase of emergence of Chill. In this period, the Team of Specialists acted independently of many organisational strategies, often only adhering to norms common to larger communities of technological practitioners. It did not fit into the customary framework of the ITU's technical wing, the CCITT. Contrary to common CCITT practices, the participants were largely computer specialists and programming language design theoreticians. The work was often organised in an ad-hoc fashion and

often, they would have to meet outside the ITU tower in Geneva. Still, the diplomatic bickering inside the Team of Specialists and the Implementors' Forum could often run along organisational boundaries.

The level of abstraction chosen as appropriate for the future language drew on computer science research, popularised through IFIP conferences and made visible through Nicklaus Wirth's PL/360 programming language. The reliability and portability concerns were also given high priority, which pays testimony to the interests of the telecommunication administrations. During the last years of the 1970s, the interests of manufacturing firms were given a clearer outing in the diplomatic process: L. M. Ericsson was to some extent able to influence the direction taken when including concurrency concepts in the language. Still, formal descriptions trumped most efforts, again pointing out how the formally oriented development virtue was activated as a norm in decision-making processes.

Chill has also to be understood as coming from a set of processes rather than the logics of a stable regime. The dissatisfaction of, in particular, the smaller state telecommunication administrations with their dependency on oligopolistic or monopolistic telecommunication manufacturers, in particular those foreign owned, spurred the initiative of standardising technologies that could allow the administration to control the procurement of equipment or the technical abilities of the equipment. Closer connections between manufacturers and administrations were sought in some markets in the 1970s, in particular in Britain, Japan and Sweden, but the impetus behind Chill was, at the outset, an alliance between administrations wishing for more control over the software in their switches. Paradoxically, this also included the Swedish telecommunication administration. However, Chill would soon be dominated by manufacturing firms, and its ultimate fate was almost completely bound up in how it was perceived by decision-makers in the dominating incumbent manufacturers of the 1980s: ITT, Siemens and Philips all used the language in the development of real switching equipment. The telecommunication administrations that had led the work abandoned Chill almost completely, with a few minor exceptions being the financial support from the Nordic administrations and the continued use of it in the Japanese administration, the NTT. L. M. Ericsson, AT&T and Northern Telecom, the three other dominant firms at the time, jumped ship, or in the latter case, they never got on board. The diffusion was tangled with technical improvements made to the language, as improvement to the modular capabilities of the language came to the fore. These improvements were mediated trough a specific Chill community, built around conferences and the publication *Chill Bulletin*. Still, these improvements were very much a result of continued support by a few manufacturing firms, in particular firms that were challenging for new international markets, like ITT and Siemens.

The long demise throughout the 1990s can best be explained as an outcome of the changing political economy of telecommunications, visible both in the organisational framework of Chill, the ITU, and in the changing relationships between manufacturers and administrations in many countries. The independence that the smaller administrations had wished for in the 1970s was won through political re-regulation. Following this, their interest in a common standard disappeared. As the close connections between administrations and manufacturers disintegrated in many larger countries, Chill also lost its institutional potency. Paradoxically, the fate of Chill in the last years of its life was intimately related to the ITU, the organisation that had so much trouble integrating Chill's Team of Specialists in the 1970s. ITU had never been a comfortable home for Chill. Still, when the ITU lost some of its powers due to the ongoing reorganisation of telecommunications, Chill faltered. The ambitions of regulating technological change through international governance were dismantled as the political economy of the sector was reframed.

The effort to gain prominence outside the realms of the "ancien regime" never came to much. The entrepreneurial efforts to spread the use of Chill beyond its initial adaptors never gained prominence and the community of Chill practitioners also disintegrated in the early 1990s. However, even just before the moment where the language was led to rest in the late 1990s, members of the Chill community were able to add object-oriented concepts to the programming language, initiated by Chill developers at Siemens. As such, the ability to develop the language still existed, even in its last rounds.

Even if it was possible to change the technology even in the 1990s, certain technical decisions hampered its ability to move beyond its initial realm. One such example was directly related to the mode of technical diplomacy that characterised the Chill project, namely the frequent use of compromises. This created some overlapping concepts in the programming language, the concepts for communication between concurrent processes being one example that I have analysed in detail. At first, one could argue that multiple features for doing (almost) the same thing could promote flexibility, and that this could ease the diffusion of the language. However, when all the large manufacturers that started using Chill could create their own subsets of the language, one ended up with a lot of firms using only portions of the language, which made entrepreneurial efforts targeting the large manufacturers difficult. This also made it less likely that those already on board the Chill ship would update to a newer version of the language, making the updates beyond 1988 more theoretical than something that was put to real use.

Furthermore, not all the desires that spurred the design and standardisation of Chill were something that was possible to achieve. Goals like increased reliability, portability of software between switches made by

different manufacturers and the possibility to add custom software to any switch, because of a common programming language, were, in some respects, ill warranted. While the reliability of programming code certainly increased by moving from assembly code to high-level languages, this was already well under way before the design of Chill. Portability did rely on a lot of other things besides a common programming language, and so did the possibility of adding software to it for outside vendors. To use a stock phrase from software development, Chill was not a silver bullet.[269]

Summing up, Chill was neither a total success nor a total failure. It was neither a radical innovation, nor just incremental change. Almost all efforts towards creating high-level programming languages for telecommunication systems can be characterised by such "betwixt-and-between-ness". This thesis has tried to explain this change by stressing the importance of organisations, norms and regimes, explanations that should go beyond the specificities of the telecommunication industry. I have highlighted how arguments that resonated with community norms and the development virtue common to many computer scientists trumped corporate strategies in the second part of the emergent phase. Throughout this thesis, I have showed how technological arguments about programmers' productivity, software reliability, code portability and program efficiency were all related to different communities of technological practitioners, and how the priorities can be understood as a process of technical diplomacy, involving different negotiators, ambassadors and loyal bureaucrats. The direction of the technological change towards high-level programming languages can therefore not be explained without taking the priorities of these participants into consideration.

## The direction of technological change

In the introductory chapter, I denounced the preoccupation with the rate of technological change evident in much literature on innovation and change. Regrettably, questions related to the direction of technological change have attracted less research. I also put forward the argument that to be able study the direction of technological change in the telecommunication sector, one would have to go beyond national specificities, technological particularities and naïve periodisation schemes. In many ways, the case study of Chill has,

[269] The expression comes from Fredrick P. Brooks legendary essay, "*No Silver Bullet – Essence and Accident in Software Engineering"*, where it is claimed that "there is no single development, in either technology or in management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity." See Frederick P. Brooks, "No Silver Bullet - Essence and Accident", in *The mythical man month: essays on software engineering* (Addision Wesley Longman, 1995).

regardless of all its peculiarities, been able to address these general concerns: its unruly becoming has highlighted the international aspects of technological decisions. Its place in time has confounded any easy periodisation scheme. I have also tried to render the technological specificities as transparent and context-free as possible, by pointing out how Chill was an example of a family of high-level languages particularly made for real-time and communication systems. This was related to the general ambitions raised in the introduction of this thesis, namely to explain how and why most organisations in the telecommunication industry started to use high-level programming languages by the 1980s in their development and production of telecommunication equipment.

In the early 1970s, programming telecommunication switches were regarded as "venturing into virtually unknown waters".[270] The question of who should be on the navigation bridge was high on the agenda in all types of telecommunication organisations, as programming was not something solely in the domain of the equipment manufacturers. In some smaller countries it was even believed that it could be something that the administrations could handle themselves. Throughout the 1970s and 1980s, different telecommunication organisations and communities of technological practitioners tried to come up with organisational and technological answers to the programming problem. One of the general answers was the use of high-level programming languages and more specifically, high-level programming languages with particular features tailored for telecommunication systems, so-called concurrent programming languages. However, this family of concurrent languages was really betwixt and between the unknown waters of the early 1970s and the more tried and tested practices of the late 1990s. By the late 1990s, these languages almost vanished, until they reappeared as important concerns to both academics and practitioners ten years later. As such, the pioneering concurrent languages were really liminal languages, but pointing towards a future reintegration.

As argued in the first few chapters of this thesis, the move towards high-level, yet specialised, programming languages in telecommunications was tied in with a general concern in both computer science and in telecommunications about how to produce reliable and efficient software for telecommunication systems. The interactions between these two domains, exemplified in chapter two with the analysis of some important conferences, were important reasons to the directions that were sought. The coincidental disappearance of many such high-level programming languages for communication systems was, in time, related to the dismantling of the "ancien regime." However, it was also marked by a shift in programming

[270] Chapuis and Joel, *Electronics, computers and telephone switching: 1960-1985*, 265.

language design, where object orientation became the attraction of many communities of technological practitioners throughout the 1980s.

On a theoretical level, this thesis has tried to explain this *direction of technological change* towards the specialised high-level languages by exploring the interplay of institutions and communities on the one hand side and strategies and organisations on the other. This has been coupled with an interest in how regimes at a political economic level meddle and direct lower level technological decision-making. Another imperative has been to stick to a denouncement of functional explanations, which implies that the change towards high-level languages could not be explained by the effects of the change, be it more reliable code, economic efficiency or more reliable software. However, that these goals can direct activities and influence community norms has been revealed throughout this thesis.

Another important theoretical objective has been to explore the limits of what can be understood as strong regime logics in the existing literature on telecommunications, the logics of the "ancien regime" if you want. In particular, I have been interested in to what extent the changes analysed followed a pattern that mirrored that of the stability of the international telecommunication regime in the first phase and its upheavals in the latter stages. When approaching technological change from the bottom up, by following the technology and processes of technical diplomacy, the limitations inherent in regime-oriented explanations became increasingly evident. In the phase of emergence, the ambassadors of computer science and those with a strong affinity towards a mathematical oriented development virtue steered the diplomatic processes, however unrelated and alien they were to the traditional domain of telephony and telecommunication engineering. In the phase of diffusion, the diplomatic processes were more unruly. As a result of the weak enforcement by the ITU, for example by not setting up a formal compilation validation system as was done with Ada, the shape and fate of the language were more open to influence of organisational strategies and local circumstance. However, the ultimate demise of the language, despite both entrepreneurial efforts and large-scale use at successful telecommunication equipment makers, cannot solely be ascribed to the internalities of the Chill project. In the end, the institutional turmoil of telecommunication policy in the early 1990s isolated Chill in its late stages of life.

All in all, it was not the stable regime of national monopolies and their prolonged international arm of the ITU that brought Chill to life, despite wishes for a programming language under the control of the regime, but it was when this regime were trembling, when it was in a transitional state, that Chill was brought to an end. One could propose that only a stable regime could have prolonged Chill's life and made it more successful: Such a complex programming language as Chill could quite simply not be widely adopted without some sort of control mechanism outside a market.

Following this, one could propose that the once dominant development virtue, influenced by ideals about the mathematically proficient programmer and affinities towards formalism, was more compatible to an institutional context where large organisations dominated, and was too complex to integrate easily into a system of small entrepreneurial start-ups. Still, it was hardly the natural offspring of such a regime.

On the individual level, Kristen Rekdal is, perhaps, the best example of how the changing conditions of Chill influenced individual actions. As a researcher turned technical diplomat and finally a business entrepreneur, Rekdal had to adapt to the changing conditions throughout the 1970s and 1980s. Rekdal had to decide about new and novel technologies, decisions that were of a different kind than "normal engineering", where technological design follows much the same patterns as Kuhn's normal science inside a paradigm. It was also deviant of what can be understood as radical inventive development, where actions are understood as contingent and highly uncertain. Rekdal made decisions that were somewhat risky, but not completely uncertain. They were made by daring technological practitioners rather than heroic inventors. The process was ambiguous but also goal-directed at its outset and depended on successful reintegration into the economic and technical system at its end. However, what it had to be reintegrated into was something that had changed. There is no better illustration of these changes than the dismantling of the international telecommunication regime. While Kristen Rekdal's move from a practically oriented research establishment such as Runit to the one of a venturing entrepreneur does, at first sight, not look all that improbable or remarkable, it might nevertheless illustrate a key point to much innovation literature: what is perceived as entrepreneurial action is really a function of its environment.

The classical perception of the Schumpeterian entrepreneur is that of a unique and creative individual who develops new products, services or techniques, and brings these developments into the economic system. Initially, the entrepreneur was perceived by Schumpeter as an individual bringing novelty into the economic system by setting up his own firm.[271] However, and this is very well known from the extensive literature on Schumpeter, his vision of the role and the importance of the entrepreneur changed over time.[272] In Schumpeter's later work, the function of the individual entrepreneur was replaced by large research laboratories, as

[271] Joseph Alois Schumpeter, *The theory of economic development; an inquiry into profits, capital, credit, interest, and the business cycle*, Harvard economic studies. (Cambridge, Mass.,: Harvard University Press, 1934).

[272] See, for example, the introduction in Richard Swedberg, *Entrepreneurship : the social science view*, Oxford management readers (Oxford ; New York: Oxford University Press, 2000).

Schumpeter's analysis of the changes in the capitalistic system underwent a major transformation.[273] If we interpret Schumpeter's writings not as a change of heart, but as a theory of how the institutional framework of entrepreneurial action changes over time (and in Schumpeter's lifetime), the question of how individuals act under liminality could add to our understanding of the crucial role of entrepreneurial action in different institutional settings. Rekdal's changing professional identities resonate with one such view, although they reveal how programming technologies moved *from* the labs to an oligopolistic market place dominated by internationally oriented incumbent manufacturers. This move was not something that was easily done for an individual entrepreneur, and in particular with a direction of technological change based on a software development virtue that was more compatible with large organisations than small start-ups. However, this was not something predetermined. The programming language Objective C, which was developed to create assistance to Chill programmers at ITT, and later spun out into a small start-up firm and then ended up at Apple and in the iPhone, highlights this unruly nature of success, technology, and institutional structures. Both Chill and Objective C were developed in periods that were essentially betwixt and between, both in an industry-specific manner and in a more general sense. In telecommunications, the 1970s and 1980s essentially led up to the break up of the "ancien regime", but it was temporarily replaced by an oligopoly of even fewer manufacturers than before. In the 1990s, this was gradually replaced by a deverticalisation, although the industry was still dominated by a few large firms. The opening up to outside vendors of programming tools illustrated this neatly, as in the case where L. M. Ericsson started using tools from the company Rational by 1997. This was also part of the general trend of vertical disintegration, specialisation and decentralisation that swept many industries in the 1990s and onwards.[274]

As proposed by the economist Richard N. Langlois, the decentralisation and deverticalisation of production was intimately dependent on the market, but also on institutions that support specialisation and exchange. One of Langlois's strongest claims is that many of these institutions take on the form of standards – and that these standards are a necessity and a cause for the change towards a market-oriented decentralised production system: "Decentralisation of production implies an ability to cut apart the stages of production cleanly enough that they can be placed into separate hands without high costs of coordination; that is to say,

[273] Joseph Alois Schumpeter, *Capitalism, socialism, and democracy* (New York, London,: Harper & Brothers, 1942).
[274] This is convincingly described and analysed in Langlois, "The vanishing hand: the changing dynamics of industrial capitalism".

decentralization implies some degree of standardization of 'interfaces' between stages."[275] One of the more peculiar characteristics of this change towards the "knowledge economy" and the decentralisation of production is the distinct character of one of its most important capital goods, namely the intangibility and elusiveness of programming languages, and their possibility to provide such interfaces between stages of production. Some of these programming languages have lived with us for a very long time, just because of their role as a standardised interface between organisations. The amount of legacy Cobol code is a very good indication of this.[276] However, as this thesis has shown, despite these characteristics programming languages are not malleable. As an effort of institutionalisation and standardisation, Chill was never able to break out completely of its initial framework, the international telecommunication regime, even though it never was closely aligned with it. This illustrates how processes of institutionalisation, like those concerned with the rules and regulations of programming embodied in programming languages, are tied in with institutions of many kinds, both regime-like structures like the ITU and with norms held by groups of technological practitioners. However, the relationship is never completely linear, nor altogether contingent.

[275] Ibid.: 374.
[276] Ensmenger, *The computer boys take over : computers, programmers, and the politics of technical expertise*.

# Appendix 1

Example of code written in Chill from *CCITT High Level Language (CHILL),* Recommendation Z.200, CCITT (1988).

```
 1  switchboard:
 2  MODULE
 3     /*  This example illustrates a swicthboard which queues incoming calls
 4         and feeds them to the operator at an even rate. Every time
 5         the operator is ready one and only one call is let through. This is
 6         handled by a call distributor which lets calls thorugh at fixed
 7         intervals. If the operator is not ready or there are other calls
 8         waiting, a new call must queue to wait for its turn. */
 9      DCL operator_is_ready,
10          switch_is_closed EVENT;
11
12      call_distributor:
13      PROCESS ();
14          wait:
15          PROC (x INT);
16              /*some wait action */
17          END wait;
18          DO FOR EVER;
19              wait(10 /*seconds*/);
20              CONTINUE operator_is_ready;
21          OD;
22      END call_distributor;
23
24      call_process:
25      PROCESS();
26          DELAY CASE
27          (operator_is_ready):/*some actions */;
28          (switch_is_closed): DO  FOR i IN INT (1:100);
29                                  CONTINUE operator_is_ready;
30                                  OD;
31          ESAC;
32      END call_process;
33
34      operator:
35      PROCESS ();
36          DCL time INT;
37          DO FOR EVER;
38              IF time = 1700
39                  THEN CONTINMUE switch_is_closed;
40              FI;
41          OD;
42      END operator;
43
44      START call_distributor();
45      START operator()
46      DO FOR i IN INT (1:100);
47          START call_process();
48      OD;
49
50   END swicthboard;
```

# Appendix 2

| | | *Co-appearance* | | *Influence* | |
|---|---|---|---|---|---|
| Name | Organization | Degree | alpha | Degree | alpha |
| C. Breeus | Philips (MBLE) | 176 | 285,25 | 891 | 3249,22 |
| R. H. Bourgonjon | Philips | 173 | 275,94 | 838 | 2941,79 |
| R. W. Meijer | PTT Netherlands | 192 | 320,28 | 697 | 2215,26 |
| K. Rekdal | RUNIT | 192 | 320,28 | 521 | 1431,63 |
| C. G. Denenberg | ITT | 84 | 136,10 | 368 | 1247,95 |
| R. Reed | GEC | 143 | 218,95 | 423 | 1113,90 |
| D. Combelic | ITT | 158 | 246,34 | 361 | 850,75 |
| H. R. Sorgenfrei | GEC | 174 | 280,41 | 335 | 749,08 |
| K. F. Clements | UKPO | 165 | 271,37 | 324 | 746,71 |
| I. Jacobson | LME | 142 | 220,30 | 296 | 663,00 |
| D. Bjørner | Tech. Univ. | 141 | 210,94 | 281 | 593,46 |
| G. Louis | Philips (MBLE) | 61 | 71,02 | 251 | 592,79 |
| K. Maruyama | NTT | 52 | 72,16 | 209 | 581,48 |
| R. Martucci | S. I. T Siemens | 192 | 320,28 | 247 | 467,33 |
| J. R. W. Smith | GEC | 34 | 34,00 | 170 | 380,13 |
| D. A. Sedar | GEC | 52 | 68,46 | 140 | 302,43 |
| O. de Bachtin | LME | 35 | 45,18 | 123 | 297,68 |
| E. Benevolo | CSELT | 159 | 248,68 | 176 | 289,61 |
| J. Devoil | ITT | 54 | 77,82 | 126 | 277,38 |
| P. W. Dell | UKPO | 86 | 132,92 | 138 | 270,19 |
| C. Langlois | CNET | 89 | 116,43 | 151 | 257,31 |
| J. Sjödin | LME | 49 | 70,01 | 115 | 251,73 |
| G. Ercolani | GTE | 89 | 116,43 | 145 | 242,13 |
| J. Aminoff | GTE | 89 | 116,43 | 145 | 242,13 |
| P. Branquart | Philips (MBLE) | 28 | 28,00 | 112 | 224,00 |
| H. D. Rovengo | ATT | 88 | 123,06 | 126 | 210,84 |
| G. Barberye | CNET | 62 | 73,60 | 124 | 208,16 |
| H. Vanooteghem | LME | 62 | 73,60 | 118 | 193,24 |
| R. Laufenburger | GTE | 62 | 73,60 | 118 | 193,24 |
| T. Denvir | ITT | 108 | 149,98 | 127 | 191,25 |

| | | | | | |
|---|---|---|---|---|---|
| D. Tann | GEC | 55 | 65,31 | 111 | 187,26 |
| P. Smith | UKPO | 53 | 61,01 | 106 | 172,56 |
| A. Rockström | PTT Sweden | 88 | 123,06 | 107 | 164,99 |
| T. Bingerfors | LME | 67 | 81,75 | 101 | 151,31 |
| R. Wirth | GEC | 28 | 28,00 | 84 | 145,49 |
| D. Cohen | ITT | 16 | 16,00 | 64 | 128,00 |
| L. Kott | CNET | 55 | 65,31 | 83 | 121,08 |
| T. Koizumi | C.I.A of Japan | 93 | 120,93 | 93 | 120,93 |
| A. Cullen | GEC | 19 | 19,00 | 57 | 98,73 |
| B. Robinet | CNET | 34 | 34,00 | 68 | 96,17 |
| S. Suzuki | LME | 34 | 34,00 | 68 | 96,17 |
| T. Sato | NTT | 34 | 34,00 | 68 | 96,17 |
| D. Jacob | ITT | 47 | 54,46 | 66 | 90,63 |
| E. Brigsted | PTT Denmark | 27 | 27,00 | 54 | 76,37 |
| N. A. Matrellotto | ATT | 27 | 27,00 | 54 | 76,37 |
| G. Mitchell | ATT | 62 | 73,60 | 62 | 73,60 |
| J. Moloney | ITT | 61 | 71,02 | 61 | 71,02 |
| D. Chappel | ATT | 49 | 70,01 | 49 | 70,01 |
| P. Neumann | PTT DDR | 52 | 57,86 | 52 | 57,86 |
| H. Kvarneby | LME | 16 | 16,00 | 32 | 45,25 |
| W. Ferreau | CSELT | 33 | 42,39 | 33 | 42,39 |
| B. Forss | Hasler | 34 | 34,00 | 34 | 34,00 |
| J. R. Rasmussen | PTT Denmark | 34 | 34,00 | 34 | 34,00 |
| K. Bryn | PTT Norway | 34 | 34,00 | 34 | 34,00 |
| P. Molnar | PTT Hungary | 34 | 34,00 | 34 | 34,00 |
| E. Camarotto | ASST | 28 | 28,00 | 28 | 28,00 |
| G. Roucairol | CNET | 28 | 28,00 | 28 | 28,00 |
| M. Ciccotti | Telettra | 28 | 28,00 | 28 | 28,00 |
| V. Giarratana | CSELT | 28 | 28,00 | 28 | 28,00 |
| G. Rochlin | PTT Australia | 27 | 27,00 | 27 | 27,00 |
| H. Nagata | KDD | 27 | 27,00 | 27 | 27,00 |
| K. Harwood | LME | 27 | 27,00 | 27 | 27,00 |
| M. Yoshioka | GEC | 27 | 27,00 | 27 | 27,00 |

| | | | | | |
|---|---|---|---|---|---|
| R. Haylock | PTT Australia | 27 | 27,00 | 27 | 27,00 |
| T. Kanda | KDD | 27 | 27,00 | 27 | 27,00 |
| D. Ritchie | ATT | 18 | 18,00 | 18 | 18,00 |
| H. Katzender | PTT Brazil | 18 | 18,00 | 18 | 18,00 |
| K. K. Basu | PTT India | 17 | 17,00 | 17 | 17,00 |
| J. D. Niessen | GEC | 16 | 16,00 | 16 | 16,00 |
| N. M. Rothon | UKPO | 16 | 16,00 | 16 | 16,00 |

## Archival sources

International Telecommunication Union Archive (ITUA), Geneva, Switzerland.

Norwegian Telecommunication Administration (NTA - Teledirektoratet in Norwegian), Norwegian National Archive, Oslo, Norway.

Two separate NTA archives have been used, one containing the material of the technical department (Teknisk avdeling / Teletjenesteavdeling in Norwegian), archive number S-2865, and the other covering the department of administration (Administrasjonsavdeling in Norwegian), archive number S-2854, Norwegian National Archives, Oslo, Norway

Norwegian Telecom Research (NTR, Teledirektoaretet, Televerkets forskningsinstitut in Norwegian) archive number S-4173, Norwegian National Archives, Oslo, Norway.

### Private collections

Kristen Rekdal (KRC), Trondheim, Norway.
Remi Bourgonjon (RBC), Heeze, the Netherlands.
Kees Smedema (KSC), Heeze, the Netherlands.

## Interviews

Remi Bourgonjon,
interview with author, 16 January 2009, Heeze, the Netherlands.
Telephone interview with author, 17 March 2011.
Emails to author, March 2011.

Richard Daley,
emails to author, April 2011.

Svein Hallsteinsen,
interview with the author, 21 January 2009, Trondheim, Norway.

Ivar Jacobson,
telephone interview with the author, 22 February 2011.

Capers Jones,
emails to author, February 2011.

Tom Love,
emails to author, February 2011.

Neil Olsen,
emails to author, February 2011.

Kristen Rekdal,
interviews with author, 28 November 2007, 8 June 2008, 11 November 2008, Oslo, Norway.
Interview with Lars Thue and Gard Paulsen, 13 September 2004, Trondheim, Norway.

Norio Sato,
emails to author, November 2008 and March 2011.

Kees Smedema,
interview with author, 20 January 2010, Heeze, the Netherlands.
Emails to author, March 2011.

## Printed sources


Abbate, Janet. *Inventing the Internet*, Inside technology. Cambridge, Mass.: MIT Press, 1999.

Acemoglu, Daron. "Directed Technical Change." *The Review of Economic Studies* 69, no. 4 (2002): 781 - 809.

Ada Language (Great Britain) Ltd. "Ada yearbook." v. London: Chapman & Hall, 1991.

Adams, Stephen B., and Orville R. Butler. *Manufacturing the future : a history of Western Electric*. Cambridge: Cambridge University Press, 1999.

Aho, Alfred V., and Jeffrey D. Ullman. *Principles of compiler design*, Addison-Wesley series in computer science and information processing. Reading, Mass.: Addison-Wesley Pub. Co., 1977.

Akera, Atsushi. *Calculating a natural world : scientists, engineers, and computers during the rise of U.S. cold war research*, Inside technology. Cambridge, Mass.: MIT Press, 2007.

———. "Voluntarism and the Fruits of Collaboration." *Technology and Culture* 42, no. 4 (2001): 710-36.

Alan, J. Perlis. "The American side of the development of Algol." *SIGPLAN Not.* 13, no. 8 (1978): 3-14.

Amesse, Fernand, Robert Latour, Claudia Rebolledo, and Louise Séguin-Dulude. "The telecommunications equipment industry in the 1990s: from alliances to mergers and acquisitions." *Technovation* 24, no. 11 (2004): 885-97.

Anchordoguy, Marie. "Japan's software industry: a failure of institutions?" *Research Policy* 29, no. 3 (2000): 391-408.

Andersen, Flemming, and Karsten Nyblad. "Compiler testing, theory and experiences." In *Third Chill Conference*, 7-12. Cambridge University: ITT Europe, 1984.

Araskog, Rand V. *The ITT wars*. 1st ed. New York: Holt, 1989.

Armstrong, Deborah J. "The quarks of object-oriented development." *Commun. ACM* 49, no. 2 (2006): 123-28.

Armstrong, Joe. "A history of Erlang." New York, NY, USA, 2007.

Badenoch, Alexander, and Andreas Fickers. *Materializing Europe : transnational infrastructures and the project of Europe*. New York: Palgrave Macmillan, 2010.

Baetjer, Howard. *Software as capital : an economic perspective on software engineering*. Los Alamitos, Calif.: IEEE Computer Society, 1998.

Balconi, Margherita, Andrea Pozzali, and Riccardo Viale. "The 'codification debate' revisited: a conceptual framework to analyze the role of tacit knowledge in economics." *Industrial and Corporate Change* 16, no. 5 (2007): 823-49.

Bauer, F. L., ed. *Advanced Course on Software Engineering*, Lecture Notes in Economics and Mathematical Systems. Berlin: Springer-Verlag, 1973.

Ben-Ari, M. *Understanding programming languages*. Chichester; New York: Wiley, 1996.

Bergin, Thomas J. "A history of the history of programming languages " *Communications of the ACM* 50, no. 5 (2007): 69-74.

Bergin, Thomas J., and Richard G. Gibson. *History of programming languages II*. New York: ACM Press; Addison-Wesley Pub. Co., 1996.

Bergmann, A., T. Letschert, and A. Lingen. "CHILL/tss – a System Development Environment for Telephone Switching Systems." In *Fifth CHILL Conference*, 201 - 09. Rio de Janeiro: Telebras, 1990.

Biancuzzi, Federico, and Shane Warden, eds. *Masterminds of Programming* Beijing: O'Reilly, 2009.

Bijker, Wiebe E. *Of bicycles, bakelites, and bulbs : toward a theory of sociotechnical change*, Inside technology. Cambridge, Mass.: MIT Press, 1995.

Bijker, Wiebe E., and John Law. *Shaping technology/building society : studies in sociotechnical change*, Inside technology. Cambridge, Mass.: MIT Press, 1992.

Binder, Hans-Eugen. "A telecommunication development: Siemens' digital switching system, EWSD." In *Proceedings of the 18th international conference on Software engineering*, 587. Berlin, Germany: IEEE Computer Society, 1996.

Bishop, J. M. *Data abstraction in programming languages*, International computer science series. Reading, Mass.: Addison-Wesley, 1986.

Bishop, R., E. Bordelon, R. Cheung, N. R. Feay, G. Louis, and C. H. Smedema. "Separate Compilation and the Development in Large Programs in CHILL." In *Fifth International Conference on Software Engineering for Telecommunication Switching Systems*, 80 - 86. Lund, Sweden: Institution of Electrical Engineers, 1983.

Bissell, Christopher. "Control in the technical societies: a brief history." *Measurement and Control* 43, no. 7 (2010): 217 - 21.

Bjørner, D., and C. B. Jones. *The Vienna development method : the Meta-language*, Lecture notes in computer science 61. Berlin: Springer-Verlag, 1978.

Bjørner, Dines. "Programming Languages: Formal Development of Interprenters & Compilers." In *International Computing Symposium*, edited by E. Morlet and D. Ribbens, 1-21. Liege, Belgium: North-Holland, 1977.

Boehm, Barry W. "Software and its Impact: A Quantitative Assessment." *Datamation* 19, no. 5 (1973): 48 - 59.

Ucinet 6 for Windows: Software for Social Network Analysis Version 6.0. Analytic Technologies, Harvard.

NetDraw: Graph Visualization Software. Version 2.097. Analytic Technologies, Harvard.

Borugonjon, Remi H. "The CCITT High Level Programming Language." In *Software Engineering for Telecommunciation Switching Systems*, 36 - 39. Helsinki, Finland: Institution of Electrical Engioneers, 1978.

———. "Programming Languages, Environments and Chill." *Chill Bulletin* 3, no. 1 (1983): 3 - 8.

Botsch, Dietrich, and Hans Eberding. "EWSD, A Real-Time Communication System with High-Level Language Software." *IEEE Transactions on Communications* 30, no. 6 (1982): 1337 - 42.

Boute, R. T., and M. I: Jackson. "A joint evaluation of the programming languages Ada and CHILL." In *Software Engineering for Telecommunciation Switching Systems*, 214 - 20. Coventry, United Kingdom: Institution of Electrical Engineers, 1981.

Brandt, Thomas, and Ola Nordal. *Turbulens og tankekraft: historien om NTNU*. Oslo: Pax, 2010.

Branquart, P., J. Lewi, M. Sintzoff, and P. L. Wodon. "The composition of semantics in Algol 68." *Commun. ACM* 14, no. 11 (1971): 697-708.

Branquart, Paul, Georges Louis, and Pierre Wodon. *An Analytical Description Of CHILL, the Ccitt High Level Language*. Edited by G. Goos and J. Hartmanis, Lecture Notes in Computer Science. Heidelberg: Springer-Verlag, 1982.

Braun, Ernest, and Stuart Macdonald. *Revolution in miniature : the history and impact of semiconductor electronics*. 2nd ed. Cambridge: Cambridge University Press, 1982.

Brooks, Frederick P. "No Silver Bullet - Essence and Accident." In *The mythical man month: essays on software engineering*, 177 - 204: Addision Wesley Longman, 1995.

Brown, J., and J. Carr III. "Automatic Programming and its Development on the MIDAC." In *Symposium on automatic programming for digital computers, 13-14 May 1954.* , edited by Mathematical Computing Advisory Panel United States Navy, 84-97. Washington: U.S. Dept. of Commerce, Office of Technical Services, 1954.

Campbell-Kelly, Martin. *From airline reservations to Sonic the Hedgehog : a history of the software industry*, History of computing. Cambridge, Mass.: MIT Press, 2003.

———. "The History of the History of Software." *IEEE Annals of the History of Computing* 29, no. 4 (2007): 40-51.

Campbell-Kelly, Martin, and Daniel D. Garcia-Swartz. "From Products to Services: The Software Industry in the Internet Era." *Business History Review* 81, no. Winter 2007 (2007): 735 - 64.

Castellacci, Fulvio. "Evolutionary and new growth theories: Are they converging?" *Journal of Economic Survey* 21, no. 3 (2007): 585-627.

Ceruzzi, Paul E. "Electronics Technology and Computer Science, 1940-1975: A Coevolution." *IEEE Annals of the History of Computing* 10, no. 4 (1989): 257-75.

———. *A history of modern computing*, History of computing. Cambridge, Mass.: MIT Press, 1998.

Chambers, David Wade, and Richard Gillespie. "Locality in the History of Science: Colonial Science, Technoscience, and Indigenous Knowledge." *Osiris* 15 (2000): 221-40.

Chapuis, Robert J., and Amos E. Joel. *Electronics, computers and telephone switching: 1960-1985*. 2 vols. Vol. 2, Studies in telecommunication 1990.

Christensen, Sverre A. "Switching Relations: The rise and fall of the Norwegian telecom industry." BI Norwegian School of Management, 2006.

Clark, M., and G. Walter. "CHILL Language Solutions for Mixed Data Formats." In *Fourth CHILL Conference*, 29 September - 2 October. Munich: Simenes AG, 1996.

Clark, Mark W., Hans-Joachim Hey, and Gerd-Arnold Schlaffke. "EWSD software modularity - smoothing the way for performance increases." In *Global Telecommunications Conference*, 1389 - 94. Hollywood, Florida: IEEE 1988.

Clark, NM., K. Neuhaus, and G. Walter. "Support Software Environment for a Multi-Processor-Development." In *Fourth CHILL Conference*, 99-106. Munich: Simenes AG, 1986.

Codding, George A. *The International Telecommunication Union; an experiment in international cooperation*. Leiden,: E. J. Brill, 1952.

Codding, George A., and Anthony M. Rutkowski. *The International Telecommunication Union in a changing world*. Dedham: Artech House, 1982.

Constant, Edward W. *The origins of the turbojet revolution*, Johns Hopkins studies in the history of technology. Baltimore: Johns Hopkins University Press, 1980.

Contractor, Farok J., and Peter Lorange. *Cooperative strategies in international business*. Lexington, Mass.: Lexington Books, 1988.

Cowan, Robin, Paul A. David, and Dominique Foray. "The Explicit Economics of Knowledge Codification and Tacitness." *Industrial and Corporate Change* 9, no. 2 (2000): 211 - 53.

Cowhey, Peter F. "The international telecommunications regime: the political roots of regimes for high technology." *International Organization* 44, no. 169-199 (1990).

Cox, Brian. "There is a silver bullet." *Byte*, 1990, 209-18.

Daley, R. W., and T. A. Haque. "ITT 1240 Digital Exchange – CHILL programming Environment." In *Second CHILL Conference*, not paginated. Lisle, Illinois, 1983.

Daston, Lorraine, and Peter Galison. *Objectivity*. New York: Zone Books, 2007.

Davies, Andrew. "The life cycle of a complex product system." *International Journal of Innovation Management* 1, no. 3 (1998): 229-56.

———. *Telecommunications and Politics: The Decentralised Alternative*. London: Pinter Publishers, 1994.

Deephouse, David L., and Mark Suchman. "Legitimacy in Organizational Institutionalism." In *The Sage Handbook of Organizational Institutionalism*, edited by Royston Greenwood, Christine Oliver, Roy Suddaby and Kerstin Sahlin-Andersson, 49 - 77. London: Sage Publications, 2008.

Denenberg, C. G. "CHILL Implementation techniques." In *Software Engineering for Telecommunciation Switching Systems*, 45 - 50. Helsinki, Finland: Institution of Electrical Engineers, 1978.

Dennis, Jack B. "Modularity." In *Advanced Course on Software Engineering*, edited by F. L. Bauer, 128 - 82. Berlin: Springer-Verlag, 1973.

Dew, Nicholas, S. Ramakrishna Velamuri, and Sankaran Venkataraman. "Dispersed knowledge and an entrepreneurial theory of the firm." *Journal of Business Venturing* 19 (2004): 659 - 79.

Diebl, Georg, Georg Schulz, and Jürgen F. H. Winkler. "Object-CHILL: The Road to Object Oriented Programming with CHILL." In *Fifth CHILL Conference*, 118 - 23. Rio de Janeiro, 1990.

Dijkstra, Edsger W. "Programming: From Craft to Scientific Discipline." In *International Computing Symposium*, edited by E. Morlet and D. Ribbens, 23-30. Liege, Belgium: North-Holland Publishing Compnay, 1977.

Dosi, Giovanni. "Technological paradigms and technological trajectories." *Research Policy* 11 (1982): 147-62.

Drake, William J. "The Rise and Decline of the International Telecommunications Regime." In *Regulating the Global Information Society*, edited by Christopher T. Marsden, 127 - 77. London: Routledge, 2000.

Däcker, Bjarne. "Concurrent Functional Programming for Telecommunications: A Case Study of Technology Introduction." Royal Institute of Technology, 2000.

Edgerton, David. *The shock of the old : technology and global history since 1900*. Oxford ; New York: Oxford University Press, 2007.

Ellevseth, Stein Erik. "The SDS Software system." In *Third CHILL Conference*, 87-92. Cambridge University, 1984.

Elster, Jon. *Explaining Technical Change*. Cambridge: Cambridge University Press, 1983.

Ende, Jan van den, Nachoem Wijnberg, and Albert Meijer. "The Influence of Dutch and EU Government Policies on Philips' Information technology Product Strategy." In *Information Technology Policy: An International History*, edited by Richard Coopey, 187 - 208. Oxford: Oxford Unievrsity Press, 2004.

Ensmenger, Nathan. *The computer boys take over : computers, programmers, and the politics of technical expertise*, History of computing. Cambridge, Mass.: The MIT Press, 2010.

Fagerberg, Jan. "Innovation: A guide to the literature." In *The Oxford Handbook of Innovation*, edited by Jan Fagerberg, David C. Mowery and Richard Nelson, 1 -26. Oxford: Oxford University Press, 2005.

Faust, Katherine. "Using Correspondence Analysis for Joint Displays of Affiliation Networks." In *Models and Methods in Social Network Analysis*, edited by Peter J. Carrington, John Scott and Stanley Wasserman, 117 - 47. Cambridge: Cambridge University Press, 2005.

Foray, Dominique. *Economics of knowledge*. Cambridge, Mass.: MIT Press, 2004.

Forss, Bertil. In *Autobiographies, "From Computing Machines to IT"*. Stockholm: National Museum of Science and Technology, Sweden, 2007.

Fransman, Martin. *Japan's computer and communications industry: the evolution of industrial giants and global competitiveness*. Oxford: Oxford University Press, 1995.

———. *The market and beyond : information technology in Japan*. Cambridge England ; New York: Cambridge University Press, 1990.

Freeman, Christopher, and Francisco Louçã. *As time goes by : from the industrial revolutions to the information revolution*. Oxford: Oxford University Press, 2001.

Freeman, Linton C. "Centrality in social networks conceptual clarification." *Social Networks* 1, no. 3 (1978): 215-39.

———. *The development of social network analysis: a study in the sociology of science*: Empirical Press, 2004.

Fridlund, Mats. "Switching Relations and Trajectories: The Development Procurement of the Swedish AXE Switching Technology." In *Public Technology Procurement and Innovation*, edited by Charles Edquist, Leif Hommen and Lena Tsipouri, 143 - 65. Norwell, Mass.: Kluwer Academic Publishers, 1999.

Friedkin, Noah E. "The development of structure in random networks: an analysis of the effects of increasing network density on five measures of structure." *Social Networks* 3, no. 1 (1981): 41-52.

Fuchs, Gerhard. "Policy-making in a system of multi-level governance-the Commission of the European Community and the restructuring of the telecommunications sector." *Journal of European Public Policy* 1, no. 2 (1994): 177 - 94.

Gorn, Saul. "Planning Universal Semi-Automatic Coding." In *Symposium on automatic programming for digital computers, 13-14 May 1954.* , edited by Mathematical Computing Advisory Panel United States Navy, 74 - 83. Washington: U.S. Dept. of Commerce, Office of Technical Services, 1954.

Griem, P. D. "Real time programming 1975: proceedings of the IFAC/IFIP Workshop Boston/Cambridge, Mass." Oxford, 1976.

Grübler, Arnulf, Nebojša Nakicenovic, and William D. Nordhaus. *Technological change and the environment*. Washington, DC: Resources for the Future ; International Institute for Applied Systems Analysis, 2002.

Hackett, Edward J., Olga Amsterdamska, Michael Lynch, and Judy Wajcman. *The Handbook of Science and Technology Studies, Third Edition*. Cambridge, Mass.: The MIT Press, 2007.

Haff, Peter. "A Formal Denition of CHILL - A Supplement to the CCITT Recommendation Z.200." In *Technical Report*. Lyngby, Denmark: Dansk Datamatik Center, 1980.

Haff, Peter, and Søren Prehn. "The TFL/DCC CHILL System Development." In *Second CHILL Conference*, Without pagination. Lisle, Illinois: Bell Laboratories, 1983.

Haigh, Thomas. "How Data Got its Base: Information Storage Software in the 1950s and 1960s." *IEEE Annals of the History of Computing*, 2009, 6-25.

Hallsteinsen, Svein. "Source level debuggers: Experience from the design and implementation of chillscope." In *Advanced Programming Environments*, edited by Reidar Conradi, Tor M. Didriksen and Dag H. Wanvik, 1-12. Trondheim: Springer, 1986.

Hammer, D., FG. Franken, and P. C. Green. "A distributed operating system for the TCP16 system." In *Fifth International Conference on SOftware Engineering for Telecommunciation Switching Systems*, 178 - 84. Lund, Sweden: Institution of Electrical Engineers, 1983.

Hamonno, F., A. Potin, R. Laeron, and C. Zampoli. "Switching System Software Base Portage to Chill." In *Fifth CHILL Conference*, edited by Antonio Palma, 272 -77. Rio de Janeiro, Brazil, 1990.

Hands, D. Wade. "The Sociology of Scientific Knowledge: Some Thoughts on the Possibilities." In *New Directions in Economic Methodology*, edited by Roger Backhouse, 75-106. London: Routledge, 1994.

Hansen, Per Brinch. "The invention of concurrent programming." In *The Origin of concurrent programming: From Semaphores to Remote Procedure Calls*, edited by Per Brinch Hansen. New York: Springer-Verlag, 2001.

Hatvany, J. "Computer languages for numerical control: proceedings of the Second IFIP/IFAC International Conference on Programming Languages for Machine Tools, PROLAMAT '73,Budapest, April 10-13, 1973." Amsterdam, 1973.

Hemdal, Göran. "AXE 10 - Software Structure and Features." *Ericsson Review* 53, no. 2 (1976): 90 - 99.

Hey, H., and K. Neuhaus. "CHILL Semaphore technique for Multiprocessing." In *Fourth CHILL Conference*, 135 - 44. Munich: Simenes AG, 1996.

Hicks, John. *The theory of wages*. London,: Macmillan, 1932.

Hill, I. D., and B. L. Meek. *Programming language standardisation*, Ellis Horwood series in computers and their applications. Chichester: Halsted Press, 1980.

Hills, M. T., and S. Kano. *Programming electronic switching systems - real-time aspects and their language implications*, IEE Telecommunications Series. Stevenage: Peter Peregrinus Ltd, 1976.

Hirvensalo, J., A. Myllkangas, and K. Rahko. "Quality standardization of telecommunciation swicthing system software." In *Software Engineering for Telecommunication Switching Systems*, 13 - 18. University of Warwick, Coventry, United Kingdom: Institution of Electrical Engineers, 1981.

Hladik, Karen J. "R&D and International Joint Ventures." In *Cooperative strategies in international business*, edited by Farok J. Contractor and Peter Lorange, 187 - 204. Lexington, Mass.: Lexington Books, 1988.

Hobday, Mike, Howard Rush, and Joe Tidd. "Innovation in complex products and system." *Research Policy* 29, no. 7-8 (2000): 793-804.

Holden, J., and A. Pink. "A Globally Optimizing CHILL Code Generator for the Motorola MC68020." In *Fourth CHILL Conference*, 235 - 44. Munich: Simenes AG, 1986.

Holmevik, Jan Rune. *Educating the machine : a study in the history of computing and the construction of the SIMULA programming language*, STS rapport ; nr 22. Dragvoll: Senter for teknologi og samfunn, Universitetet i Trondheim, 1994.

———. *Inside innovation: The Simula Research Laboratory and the History of the Simula Programming Language*. Oslo: Simula Research Laboratory, 2004.

Hughes, Thomas Parke. *Networks of power : electrification in Western society, 1880-1930*. Baltimore: Johns Hopkins University Press, 1983.

*IEEE Annals of the History of Computing*. Vol. 6, 1984.
*IEEE Annals of the History of Computing*. Vol. 7, 1985.
"International Switching Symposium." Kyoto, October 25-29 1976.
Iriye, Akira. *Global Community: The Role of International Organizations in the Making of the Contemporary World*. Berkeley: University of California Press, 2002.
Jacobson, Harold K. "ITU: A potpurri of Bureaucrats and Industrialists." In *The Anatomy of Influence*, edited by Robert Cox and Harold K. Jacobson, 59 - 101. New Haven, Conn.: Yale University Press, 1973.
Jacobson, Ivar. "Concepts for Modeling Large Real Time Systems." The Royal Institute of Technology, 1985.
Johnson, Ann. *Hitting the brakes : engineering design and the production of knowledge*. Durham: Duke University Press, 2009.
Johnson, Björn, Edward Lorenz, and Bengt-Åke Lundvall. "Why all this fuss about codified and tacit knowledge." *Industrial and Corporate Change* 11, no. 2 (2002): 245-62.
Johnson, D. R., and C. P. Miller. "Testing a CHILL Compiler." In *Fourth CHILL Conference*, 209 - 18. Munich: Siemens AG, 1986.
Kakuma, M., K. Maruyama, and T. Koizumi. "DPL-A High Level Programming Language for Electronic Switching Systems." In *International Switching Symposium*. Kyoto, 1976.
Kaplan, David. "State Policy and Technological Change-The Development of the South African Telecommunications Industry." *Journal of Southern African Studies* 15, no. 4 (1989): 565-80.
Karlsson, Sven Olof, and Anders Lugn. *Changing the world : the story of Lars Magnus Ericsson and his successors*. Stockholm: Sellin & partner, 2009.
Katzeff, Kurt, and Anders Rickström. "Software standards in the field of telecommunications." In *Fifth International Conference on Software Engineering for Telecommunication Switching Systsems*, 233.36. Lund, Sweden: Institution of Electrical Engineers, 1983.
Keister, W., R. W. Ketchledge, and H. E. Vaughan. "No. 1 ESS: System Organization and Objectives." *Bell System technical Journal* 43, no. 5 (1964): 1831 - 44.
Klepper, Steven. "Entry, Exit, Growth, and Innovation over the Product Life Cycle." *The American Economic Review* 86, no. 3 (1996): 562-83.
Knight, Frank H. *Risk, uncertainty and profit*. Boston and New York,1921.
Krige, John, and Kai-Henrik Barth. *Global power knowledge: science and technology in international affairs*. Chicago, Ill.: University of Chicago Press, 2006.
Kronental, M., J. W. Roberts, K. H. Timmesfeld, and I. C. Wand. "The LTPL-E tasking proposals." *Software: Practice and Experience* 11, no. 1 (1981): 85-97.

Kuhn, Thomas S. *The structure of scientific revolutions*. 2nd ed. Chicago,: University of Chicago Press, 1970.

Langlois, Richard N. "Knowledge, Consumption, and Endogenous Growth." *Journal of Evolutionary Economics* 11, no. 1 (2001): 77-93.

———. "The vanishing hand: the changing dynamics of industrial capitalism." *Industrial and Corporate Change* 12, no. 2 (2003): 351-85.

Langlois, Richard N., and W. Edward Steinmueller. "The Evolution of Competetive Advantage in the Worldwide Semiconductor industry, 1947 - 1996." In *Richard R. Nelson*, edited by David C. Mowery, 19 - 78. Cambridge: Cambridge University Press, 1999.

Larsen, Peter Gorm. "The VDM Bibliography." Odense: The Institute of Applied Computer Science, 1996.

Larsson, Lars-Göran. "Future Telecommunications in Japan - Policy and Technology." In *Utlands rapport från Sveriges Tekniska Attachéer*. Stockholm: Sveriges Tekniska Attachéer, 1984.

Latour, Bruno. *Science in action : how to follow scientists and engineers through society*. Cambridge, Mass.: Harvard University Press, 1987.

Layton, Edward T. "Technology as Knowledge." *Technology and Culture* 15, no. 1 (1974): 31-41.

Leslie, William Henderson Paterson. "Numerical control programming languages: proceedings of the 1st International IFIP/IFAC PROLAMAT Conference, Rome 1969." Amsterdam, 1970.

Lipartito, Kenneth. "Rethinking the invention factory: Bell Laboratories in Perspective." In *The Challange of Remaining Innovative*, edited by Sally H. Clarke, Naomi R. Lamoreaux and Steven W. Usselman, 132 - 62. Stanford, California: Stanford Business Books, 2009.

Love, Tom. *Object lessons : lessons learned in object-oriented development projects*, Advances in object technology 1. New York: SIGS Books, 1993.

Lundin, Per. "Tidlig programmering : Transkript av ett vittnesseminarium vid Tekniska museet i Stockholm den 16 mars 2006." 43. Stockholm: Filosofi och teknikhistoria, 2007.

MacKenzie, Donald A. "A View from the Sonnenbichl: On the Historical Sociology of Software and System Dependendability." In *History of Computing: Software Issues*, edited by Reinhard Keil-Salwik Ulf Hashagen, Arthur L. Norberg, 97-122. Berlin: Springer, 2002.

———. *Mechanizing proof : computing, risk, and trust*, Inside technology. Cambridge, Mass.: MIT Press, 2001.

Magneli, Peter. "Communications Computer APN 167 with ERIPASCAL." *Ericsson Review* 63, no. 4 (1986): 165 - 69.

Mahoney, Michael S. "Finding a history for software engineering." *Annals of the History of Computing* 26 (2004): 8-19.

———. "The histories of computing(s)." *Interdisciplinary Science Reviews* 30, no. 2 (2005): 119-35.
———. "Software as Science - Science as Software." In *History of Computing: Software Issues*, edited by Reinhard Keil-Salwik Ulf Hashagen, Arthur L. Norberg. Berlin: Springer Verlag, 2002.
———. "Software: The Self-Programming Machine." In *From 0 to 1: An Authoritative History of Modern Computing*, edited by Atsushi Akera and Frederik Nebeker, 91 - 100. Oxford: Oxford University Press, 2002.
———. "What Makes the History of Software Hard." *IEEE Annals of the History of Computing*, 2008, 8-18.
Mangold, Werner B. "N/C Language Standardization in I.S.O." In *The Second IFIP/IFAC International Conference on Programming Languages for Machine Tools, PROLAMAT '73*, edited by J. Hatvany, 243 - 75. Budapest: North-Holland Publishing Company, 1973.
Manninen, Ari T. "Elaboration on NMT and GSM Standards." Univeristy of Jyväskylä, 2003.
Martin, M. "Utilization of the high level language Pape for the E12 siwtching system software." In *Third International Conference on Software Engineering for telecommunciation Switching Systems*, 173 - 76. Helsinki, Finland: Institution of Electrical Engineers, 1978.
Maruyama, K., N. Sato, and K. Konishi. "NTT CHILL implementation aspects and its application experience." In *Software Engineering for Telecommunciation Switching Systems*, 191-96. University of Warwick, Coventry: Institution of Electrical Engineers, 1981.
Marx, Leo. "Technology - the Emergence of a Hazardous Concept." *Technology and Culture* 51, no. 3 (2010): 561 - 77.
Mazor, Stanley. "Intel's 8086." *IEEE Annals of the History of Computing* 32, no. 1 (2010): 75-79.
Mehner, T., R. Tobiasch, and J. F. H. Winkler. "A Proposal for an Integrated Programming Environment for CHILL." In *Third CHILL Conference*. Cambridge University: ITT Europe, 1984.
Mehner, T., and J. F. H. Winkler. "An Implementation of the New CHILL-I/O." In *Third CHILL Conference*, 61 - 64. Cambridge University: ITT Europe, 1984.
Meijer, R. W., and G. H. te Sligte. "Status report of CCITT HLL implementation at the Dr Neher Laboratory of the Netherlands PTT." In *Software Engineering for Telecommunciation Switching Systems*, 51 - 55. Helsinki, Finland: Institution of Electrical Engineers, 1978.

Meiling, Erik, and Steen U. Palm. "A Comparative Study of CHILL and Ada on the Basis of Denotational Descriptions." In *Second CHILL Conference*, Without pagination. Lisle, Illinois: Bell Laboratories, 1983.

Meyer, John W., and Brian Rowan. "Institutionalized organizations: Formal structure as myth and ceremony." *American Journal of Sociology* 83, no. 2 (1977): 340 - 63.

Meyer, Peter. "A CHILL-based Systems Development for BIGFON." In *Third CHILL Conference*, 61 - 64. Cambridge University: ITT Europe, 1984.

———. "Process Communication in a CHILL Environment." In *Fourth CHILL Conference*, 25 - 30. Munich: Simenes AG, 1986.

Mirowski, Philip, and Esther-Mirjam Sent. *Science bought and sold : essays in the economics of science*. Chicago: University of Chicago Press, 2002.

Mokyr, Joel. *The gifts of Athena : historical origins of the knowledge economy*. Princeton, [N.J.]: Princeton University Press, 2002.

Mowery, David C. *The international computer software industry : a comparative study of industry evolution and structure*. New York: Oxford University Press, 1996.

Mustar, Philippe, and Philippe Larédo. "Innovation and research policy in France (1980-2000) or the disappearance of the Colbertist state." *Research Policy* 31, no. 1 (2002): 55-72.

Naur, Peter, Brian Randell, J. N. Buxton, and NATO Science Committee. *Software engineering : concepts and techniques : proceedings of the NATO conferences*. New York: Petrocelli/Charter, 1976.

Nelson, Richard R., and Sidney G. Winter. *An evolutionary theory of economic change*. Cambridge, Mass.: Belknap Press of Harvard University Press, 1982.

Nightingale, Paul. "If Nelson and Winter are only half right about tacit knowledge, which half? A Searlean critique of 'codification'." *Industrial and Corporate Change* 12, no. 2 (2003): 149-83.

Nilsson, Mikael. "Staten och kapitalet: Betydelsen av det dynamiska samspelet mellan offentligt och privat för det svenska telekomundret : Transkript av ett vittnesseminarium vid Tekniska museet i Stockholm den 18 mars 2008." 46, 2008.

Noam, Eli. "International Telecommunications in Transition." In *Changing the Rules: Technological Change, International Competition, and Regulation in Communications*, edited by Robert W. Crandall and Kenneth Flamm, 257 - 97. Washington, D.C.: The Brookings Institution, 1989.

Nofre, David. "Unraveling Algol: US, Europe, and the Creation of a Programming Language." *IEEE Annals of the History of Computing* 32, no. 2 (2010): 58 - 68.

Nordal, Ola. *Verktøy og vitenskap: datahistorien ved NTNU*. Trondheim: Tapir akademisk, 2010.
North, Douglass C. *Institutions, institutional change and economic performance*, The Political economy of institutions and decisions. Cambridge: Cambridge University Press, 1990.
OECD. "Telecommunications Equipment: Changing Markets and Trade Structures, No. 2." In *OECD Digital Economy Papers*: OECD Publishing, 1991.
Oldfield, H.R. "General Electric enters the computer business-revisited." *IEEE Annals of the History of Computing* 17, no. 4 (1995): 46 - 55.
Olsen, Odd Einar, and Ole Andreas Engen. "Technological change as a trade off between social construction and technological paradigms." *Technology in Society* 29 (2007): 456 - 68.
Oosterwijk, Herman. "Switching Technology through Five Decades: Dutch Telecommunications under Change." In *Buidling bridges between ideas and markets*, edited by Frans van Waarden, 75 - 138, 2002.
Opsahl, Tore. *Structure and Evolution of Weighted Networks*: University of London (Queen Mary College), London, UK, 2009.
Opsahl, Tore, Filip Agneessens, and John Skvoretz. "Node centrality in weighted networks: Generalizing degree and shortest paths." *Social Networks* 32, no. 3 (2010): 245-51.
Ostrom, Elinor. *Governing the commons : the evolution of institutions for collective action*, The Political economy of institutions and decisions. Cambridge: Cambridge University Press, 1990.
Owen, Geoffrey. *From Empire to Europe*. London: Harper Collins, 1999.
Parker, David. *The Official History of Privatisation Vol. I: The formative years 1970-1987*: Routledge, 2009.
Parnas, David Lorge. "On the Criteria To Be Used in Decomposing Systems into Modules." *Commun. ACM* 15, no. 12 (1972): 1053-58.
Paulsen, Gard. "Samarbeidets protokoll: utviklingen av et nordisk datanett, 1971 - 1981." G. Paulsen, 2004.
Peter, Naur. "The European side of the last phase of the development of ALGOL 60." *SIGPLAN Not.* 13, no. 8 (1978): 15-44.
Pink, A. "Fault Correction in a Running CHILL System." In *Fifth CHILL Conference*, 129 - 36. Rio de Janeiro: Telebras, 1990.
Poel, W. L. van der, L. A. Maarssen, and International Federation for Information Processing. Technical Committee 2. *Machine oriented higher level languages : proceedings of the IFIP Working Conference on Machine Oriented Higher Level Languages, Trondheim, Norway, August 27-31, 1973*. Amsterdam; New York: North-Holland Pub. Co. ; American Elsevier, 1974.

Pontarollo, Enzo. "Procurement and market structure in the telecommunications industry : A European survey." *European Journal of Purchasing & Supply Management* 1, no. 2 (1994): 88-97.

Priestley, Peter Mark. "Logic and the development of programming languages, 1930 - 1975." University College London, 2008.

Radin, George. "The early history and characteristics of PL/I." *ACM SIGPLAN Notices* 13, no. 8 (1978): 227-41.

Rain, Mark. "Some formal language aspects of Mary or Algol X revistited." *Algol Bulletin*, 1972.

Reithmaier, Erwin. "Compilation Control in a Large CHILL Application." In *Second CHILL Conference*. Lisle, Illinois: Bell Laboratories, 1983.

Rekdal, Kristen. "CHILL - The International Standard Language for Telecommunications Programming." *Telektronikk,* 89, no. 2/3 (1993): 5-10.

———. "CHILL - The Standard language for Programming SPC Systems." *IEEE Transactions on Communications* 30, no. 6 (1982): 1318-28.

———. "The Nordic CHILL Project." In *Runit Report*. Trondheim: Runit, 1980.

Ritchie, Dennis M. "The Development of the C Language." In *History of programming languages II*, edited by Thomas J. Bergin and Richard G. Gibson, 671 - 98 New York; Reading, Mass.: ACM Press; Addison-Wesley Pub. Co., 1996.

Robinson, Hugh, and Helen Sharp. "The emergence of object-oriented technology: the role of community." *Behaviour & Information Technology* 28, no. 3 (2009): 211-11.

Romer, Paul M. "Endogenous Technological Change." *Journal of Political Economy* 98 (1990): 71 - 102.

Rosenberg, Nathan. *Inside the black box : technology and economics*. Cambridge [Cambridgeshire] ; New York: Cambridge University Press, 1982.

———. "Technological Change in the Machine Tool Industry, 1840 - 1910." *The Journal of Economic History* 23, no. 4 (1963): 414-43.

Rowe, Mary J. "Interfacing Chill with existing C-based systems." In *Third CHILL Conference*, 27 - 33. Cambridge University: ITT Europe, 1984.

Rudmik, A., and B. G. Moore. "The Seperate Compilation of Very Large CHILL Programs." In *Second CHILL Conference*. Lisle, Illinois: Bell Laboratories, 1983.

Russell, Andrew L. ""Industrial Legislatures": Consensus Standardization in the Second and Third Industrial Revolutions." The Johns Hopkins University, 2007.

———. "Standardizing in History: A Review Essay with an Eye to the Future." In *The Standards Edge: Future Generations*, edited by Sherrie Bolin, 247 - 60. Ann Arbor: Sheridan Press, 2005.

Ruttan, Vernon W. "Induced Innovation, Evolutionary Theory and Path Dependece: Sources of Techncial Change." *The Economic Journal* 107, no. 444 (1997): 1520-29.

———. *Technology, growth, and development : an induced innovation perspective*. New York: Oxford University Press, 2001.

———. "Usher and Schumpeter on Invention, Innovation, and Technological Change." *The Quarterly Journal of Economics* 73, no. 4 (1959): 596-606.

Ryan, J. S. "Signalling and Switching as we enter the second century." *Telecommunication Journal* 43, no. 3 (1973): 206 - 19.

Ryder, Barbara G., Mary Lou Soffa, and Margaret Burnett. "The Impact of Software Engineering Research on Modern Programming Languages." *ACM Transactions on Software Engineering and Methodology* 14, no. 4 (2005): 431-77.

Sakamura, Ken. "The TRON Project." *Information and Software Technology* 38, no. 3 (1996): 239-51.

Sammet, Jean E. *Programming languages: history and fundamentals*, Prentice-Hall series in automatic computation. Englewood Cliffs, N.J.,: Prentice-Hall, 1969.

———. "Why Ada is not just another programming language." *Commun. ACM* 29, no. 8 (1986): 722-32.

Schatzenberg, Eric. "Technik Comes to America: Changing Meanings of Technology before 1930." *Technology and Culture* 47, no. 3 (2006): 486 - 512.

Schefer, J., J. Schiffer, and J. Weiser. "A Machine Independent Model for Flexible Construction of CHILL Code Generators." In *Fifth CHILL Conference*, 32 - 40. Rio de Janeiro: Telebras, 1990.

Schlaffke, G. A., J. Lantermann, and G. Becker. "A CHILL Procedure Tracer For a Real Time Multiprocessor Environment." In *Fifth CHILL Conference*, 297 - 304. Rio de Janeiro: Telebras, 1990.

Schmidt, Susanne K., and Raymund Werle. *Coordinating technology : studies in the international standardization of telecommunications*, Inside technology. Cambridge, Mass.: MIT Press, 1998.

Schumpeter, Joseph Alois. *Business cycles; a theoretical, historical, and statistical analysis of the capitalist process*. 1st ed. New York, London,: McGraw-Hill Book Company, inc., 1939.

———. *Capitalism, socialism, and democracy*. New York, London,: Harper & Brothers, 1942.

———. *The theory of economic development; an inquiry into profits, capital, credit, interest, and the business cycle*, Harvard economic studies. Cambridge, Mass.,: Harvard University Press, 1934.

Scrotesse, A. "OO_CHILL: Integrating the object paradigm into CHILL." In *Fifth CHILL Conference*, 111- 17. Rio de Janeiro, 1990.

Shane, Scott Andrew. *Academic entrepreneurship : university spinoffs and wealth creation*, New horizons in entrepreneurship. Cheltenham, UK ; Northampton, MA: E. Elgar, 2004.

Slaton, Amy, and Janet Abbate. "The Hidden Lives of Standards: Technical Prescriptions and the Transformation of Work in America." In *Technologies of power: essays in honor of Thomas Parke Hughes and Agatha Chipley Hughes*, edited by Michael Thad Allen and Gebrielle Hecht, 95 - 143. Cambridge, Mass.: MIT Press, 2001.

Sligte, G. H. te. "A cross-implementation of Chill on existing hardware under a commercial operating system." In *Software Engineering for Telecommunciation Switching Systems*, 197 - 201. University of Warwick, Coventry: Institution of Electrical Engineers, 1981.

———. "A programming environment for Chill." In *Second CHILL Conference*, Without pagination. Lisle, Illinois: Bell Laboratories, 1983.

Smedema, C. H. "CHILL: Facilities for Concurrency." In *COMPSAC*. Chicago: IEEE Computer Society Press, 1983.

———. "Real time programming, 1977: proceedings of the IFAC/IFIP Workshop, Eindhoven, Netherlands, 20-22 June 1977." Oxford, c1978.

———. "Some Issues in the International Standardization of CHILL and Ada." *Computers & Standards* 4, no. 2 (1985): 95-100.

Smedema, C. H., P. Medema, and M. Boasson. *The programming languages : Pascal, Modula, CHILL, and Ada*. Englewood Cliffs, N.J.: Prentice/Hall International, 1983.

Smith, Peter J. "Experiences in Achiving A Full Implementation of CHILL." In *Fourth CHILL Conference*, 245-49. Munich: Siemens AG, 1986.

———. "Experiences in achiving a full implementation of Chill." In *Fourth CHILL Conference*, 245-49. Munich: Siemens AG, 1996.

Sobel, Robert. *I.T.T. : the management of opportunity*. New York, N.Y.: Times Books, 1982.

"Software Engineering." Garmisch, Germany, 7 - 11 October 1968.

"Software Engineering for Telecommunication Switching Systems." Stevenage, 2 - 5 April 1973.

"Software Engineering Techniques." Rome, 27 - 31 October 1969.

Sogner, Knut. *En liten brikke i et stort spill : den norske IT-industrien fra krise til vekst 1975-2000*. Bergen: Fagbokforl., 2002.

Star, Susan Leigh, and James R. Griesemer. "Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39." *Social Studies of Science* 19, no. 3 (1989): 387-420.

Steinmueller, W. Edward. "The European software sectoral system of innovation." In *Sectoral Systems of Innovation: Concepts, Issues and Analyses of Six Major Sectors in Europe*, edited by Franco Malerba. Cambridge: Cambridge University Press, 2004.

———. "The US Software Industry: An Analysis and Interpretive History." In *International Computer Software Industry*, edited by David C. Mowery. Oxford: Oxford University Press, 1996.

Stroustrup, Bjarne. "Evolving a language in and for the real world: C++ 1991-2006." In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, 4-1-4-59. San Diego, California: ACM Press, 2007.

———. "A history of C++: 1979 - 1991." *ACM SIGPLAN Notices* 28, no. 3 (1993): 271 - 98.

Swedberg, Richard. *Entrepreneurship : the social science view*, Oxford management readers. Oxford ; New York: Oxford University Press, 2000.

"Switching Techniques for Telecommunication Networks." London, 1969.

Takahashi, Takuma, and Fujio Namiki. "Three attempts at "de-Wintelization": Japan's TRON project, the US government's suits against Wintel, and the entry of Java and Linux." *Research Policy* 32, no. 9 (2003): 1589-606.

Takamura, Shinji, Hiroshi Kawashima, Hajime Nakajima, and M. T. Hills. *Software design for electronic switching systems*, IEE telecommunications series 8. Stevenage Eng. ; New York: P. Peregrinus on behalf of the Institution of Electrical Engineers, 1979.

Tatarchenko, Ksenia. "Cold War Origins of the International Federation for Information Processing." *IEEE Annals of the History of Computing* 32, no. 2 (2010): 46-57.

Team, R Development Core. Vienna, Austria: R Foundation for Statistical Computing, R: A Language and Environment for Statistical Computing.

Thue, Lars. "Norway: a resource-based and democratic capitalism." In *Creating Nordic Capitalism: The business history of a competetive periphery*, edited by Susanna Fellman, Martin Jes Iversen, Hans Sjögren and Lars Thue, 394 - 493. Basingstoke: Palgrave MacMillan, 2008.

———. "Norway: a resource-based and democratic capitalism." In *Creating Nordic capitalism: the business history of a competetive periphery*, S. 394-493. Basingstoke: Palgrave Macmillan, 2008.

———. *Nye forbindelser: 1970-2005*. Oslo: Gyldendal, 2006.

Tiemann, Michael. "The future of Cygnus Solutions : and entrepreneur's account." In *Open sources : voices from the open source revolution*, edited by Chris DiBona, Sam Ockman and Mark Stone, 71 - 90. Beijing ; Sebastopol, CA: O'Reilly, 1999.

Turner, Victor Witter. *The forest of symbols; aspects of Ndembu ritual*. Ithaca, N.Y.,: Cornell University Press, 1967.

United States Navy, Mathematical Computing Advisory Panel. *Symposium on automatic programming for digital computers, 13-14 May 1954.* . Washington,: U.S. Dept. of Commerce, Office of Technical Services, 1955.

Utterback, James M., and William J. Abernathy. "A dynamic model of process and product innovation." *Omega* 3, no. 6 (1975): 639-56.

Vaughan, E. Earle. "Development history of No. 1 ESS - Software." In *Switching Techniques for Telecommunication Networks*, 475 - London, 1969.

Vedin, Bengt-Arne. *Teknisk revolt: Det svenska AXE-systemets brokiga framgångshistoria* Stockholm: Atlantis, 1992.

Vries, Marc de. *80 years of research at the Philips Natuurkundig Laboratorium (1914-1994): the role of the Nat.Lab. at Philips*. Edited by Kees Boersma. Amsterdam: Pallas Publications, 2005.

Walk, Kurt. "Roots of Computing in Austria: Contributions of the IBM Vienna Laboratory and Changes of Paradigms and Priorities in Information Technology." In *Human choice and computers: Issues of Choice and QUality of Life in the Information Society*, edited by Klaus Brunnstein and Jacques Berleur, 77-87. Dordrecht: Kluwer Academic Publishers, 2002.

Walker, William. "Entrapment in large technology systems: institutional commitment and power relations." *Research Policy* 29, no. 7-8 (2000): 833-46.

Wallenstein, Gerd. *Setting Global Telecommunication Standards: The Stakes, The Players & The Process*. Norwood, MA: Artech House, 1990.

Wen, W. "Problem Oriented Languages." In *Second CHILL Conference*, not paginated. Lisle, Illinois, 1983.

Wenger, Etienne. *Communities of practice : learning, meaning, and identity*, Learning in doing. Cambridge, U.K. ; New York, N.Y.: Cambridge University Press, 1998.

Wexelblat, Richard L. *History of programming languages*. New York: Academic Press, 1981.

Whitaker, William A. "Ada—the project: the DoD high order language working group." *ACM SIGPLAN Notices* 28, no. 3 (1993): 299-331.

Williams, Theodore J. "CAM and NC Software Systems: Needs for and Benefits From Generalized and Multi-Industry Standardized Languages." In *The Second IFIP/IFAC International Conference on Programming Languages for Machine Tools, PROLAMAT '73*, edited by J. Hatvany, 1 - 29. Budapest: North-Holland Publishing Company, 1973.

Winkler, J. F. H. "The Realization of Data Abstractions in CHILL." In *Third CHILL Conference*, 175 - 82. Cambridge University: ITT Europe, 1984.

Winkler, Jurgen. "A New Methodology for I/O and its Application in CHILL." In *Second CHILL Conference*. Lisle, Illinoise: Bell Laboratories, 1983.

Winkler, Jürgen F. H. "CHILL 2000." *Telektronikk*, no. 4 (2000): 70 - 77.

Wirth, Niklaus. "A Brief History of Software Engineering." *IEEE Annals of the History of Computing* 30, no. 3 (2008): 32 - 39.

———. *Compiler construction*, International computer science series. Harlow, England ; Reading, Mass.: Addison-Wesley Pub. Co., 1996.

———. "On the Design of Programming Languages." Information Processing 74, Stockholm, Sweden, August 5-10 1974.

Yates, JoAnne. *Structuring the information age : life insurance and technology in the twentieth century*, Studies in industry and society. Baltimore, Md.: Johns Hopkins University Press, 2005.

Yong Rai, Kwon. "Software technology and industry of Korea: widening horizon and emerging presence." Orlando, FL, USA.