This is the accepted manuscript of: Walrave, B., Dolmans, S., van Oorschot, K. E., Nuijten, A. L. P., Keil, M., & van Hellemond, S. (2022). Dysfunctional Agile–Stage-Gate Hybrid Development: Keeping Up Appearances. International Journal of Innovation and Technology Management, 19(03), 2240004. https://doi.org/10.1142/S0219877022400041

DYSFUNCTIONAL AGILE–STAGE-GATE HYBRID DEVELOPMENT: KEEPING UP APPEARANCES

Increasingly, the development of today's "smart" products requires the integration of both software and hardware in embedded systems. To develop these, hardware firms typically enlist the expertise of software development firms to offer integrated solutions. While hardware firms often work according to a plan-driven approach, software development firms draw on Agile development methods. Interestingly, empirically little is known about the implications and consequences of working according to contrasting development methods in a *collaborative* project. In response to this research gap, we conducted a process study of a collaborative development project involving a software firm and a hardware firm, within which the two firms worked according to contrasting development methods. We found that the software firm was gradually compelled to forgo its Agile method, creating a role conflict in terms of its way of working. As such, our results contribute to the literature on Agile–Stage-Gate hybrids by demonstrating how, in collaborative embedded systems development, hybridization of development methods may cause projects to fail. Our main practical implication entails the introduction of the "sequential Agile approach."

1. Introduction

Products are becoming more complex, and as a result their development is also becoming increasingly challenging (Kaisti et al., 2013; Tura et al., 2017; Kortelainen and Lättilä, 2013; Vatananan and Gerdsri, 2012). Notably, many of today's new smart products, such as smartphones, navigation tools, cars, and even home appliances, involve intricate software and hardware. Interorganizational collaborations may serve as a means to cope with the demands underlying the development of complex embedded systems (Das and Teng, 2000), as different actors can bring in unique and complementary knowledge and skills (Bstieler, 2005). Indeed, collaborations in the form of, for example, supplier involvement (Van Echtelt et al., 2008) have been found to improve focal firms' innovative capability and the success of new product development (Faems et al., 2005; Peng et al., 2012; Van Echtelt, 2008). This may explain why integrated software and hardware are frequently developed by highly specialized cooperating firms.

Typically, software and hardware firms utilize different development strategies. In answer to the particularities of coding complexity, software developers have shifted from plan-driven product development methods to fundamentally different Agile approaches[1] (e.g., Scrum, XP) (Ågerfalk, et al. 2009; Conboy, 2009; Fowler and Highsmith, 2001; Paluch et al., 2020; Annosi et al., 2020). While this transition appears as a logical evolutionary step from a software development perspective, it breaks with the long-dominant plan-driven (or "traditional") method, still favored and maintained by many hardware developers. As such, the differences between the two development methods may give rise to conflict in collaborative development projects in which software and hardware development must go hand-in-hand.

Scholars have considered the integration or hybridization of Agile and plan-based methods. Several studies illustrate, for instance, that elements of an Agile approach may effectively be integrated into a plan-based development method (Cooper, 2008; Datar et al., 1997; Cooper and Sommer, 2016; Edwards et al., 2019; Port and Bui, 2009) or vice versa (Port and Bui, 2009; Vigden and Wang, 2009; Bianchi et al., 2020). But such insights are derived from single-company contexts, where one method is *modified* to include elements of the other. Other studies have suggested that Agile and plan-driven methods may effectively be combined through a process called "modularization" (e.g., Lenfle and Loch, 2010; Baldwin and Clark, 2006), in which a project is divided into independent modules (Baldwin and Clark, 2006; Ghezzi and Cavallo, 2020) that are then handled by autonomous product development teams that can use their own preferred development approach (Austin and Devin; 2009; Lenfle and Loch, 2010). While the decomposition of a complex system into discrete subsystems with loose coupling might indeed allow for particular projects to be handled more efficiently (Baldwin and Clark, 2006; D'Adderio and Pollock, 2014), it seems a less suitable development strategy when there is significant interdependence between the different components (Peng et al., 2014), because it might be impossible to truly create independent subsystems. In this respect, in the context of *collaborative* embedded systems development, software and hardware are likely developed by different highly specialized firms that need intense cross functional and interfirm collaboration to achieve success (Peng et al., 2014).

[1] E.g., in software development, product requirements are very likely to change (significantly) over the course of a project, making for an ill fit with the core idea of plan-driven development that system requirements can be reliably established up front.

We seek to address this gap in the literature and uncover the implications and consequences of working according to contrasting development methods (i.e., Agile vs. plan-based methods) in a collaborative embedded systems development project. In fact, very little is known about such a collaborative development context, in which diverging development methods are maintained. In response, we conducted an in-depth case study of a collaborative embedded systems development project, in which an experienced "Agile" software development firm was contracted by a "traditional" incumbent hardware manufacturer. Drawing on a process research approach and systems thinking (Langley, 1999; Sterman, 2000; Van de Ven et al., 2000), we uncovered self-destructive dynamics that resulted from the use of conflicting development methods. In particular, we found that the software firm was gradually compelled to forgo its Agile method, creating a role conflict regarding its way of working. Consequently, the software firm resorted to "keeping up appearances" (complying with plan-based milestones while trying to maintain Agile development) to please the hardware firm, even though doing so undermined the feasibility and quality of the entire project. As a result, by forcing an Agile supplier to comply with plan-driven demands, the manufacturer sabotaged its own development.

Our findings thus show that hybridization of development methods may lead to project failure in collaborative embedded systems development. Specifically, our findings contribute to the literature on hybrid development tactics that has outlined *functional hybrid forms* (i.e., successful Agile-plan-based integration within a single company context) (e.g., Cooper and Sommer, 2016; Edwards et al., 2019) by pointing to a *dysfunctional hybrid* mode, which manifests in a collaborative context and is driven by keeping up appearances. The formalization of such dysfunctional hybridization serves to illustrate boundary conditions for Agile-plan-based integration and provides a basis for critical reflection on studies that argue that such integration may readily be achieved (Cooper and Sommer, 2016; Edwards et al., 2019). From a practical perspective, based on the rich case data and insights, we propose a new project management technique that we call the "sequential Agile approach" and that we developed specifically to mitigate the potentially vicious keeping up appearances process.

2. Plan-Driven and Agile Development in a Collaborative Context

Arguably, contemporary project management originated from the Manhattan Project, which developed the first atomic bomb in the 1940s, followed by the Atlas and Polaris ballistic missile projects of the 1950s (Lenfle and Loch, 2010; Meredith and Mantel, 2011). These projects paved the way for what we now refer to as "traditional" or "plan-driven" development. This project management technique emphasizes project control and uncertainty reduction and is actionable by tools such as PERT, CPM, and the Gantt chart. Even though this development tradition is now well over half a century old, it is still widely used, especially for developing physical products (Cooper and Sommer, 2016).

A distinctive feature of plan-driven development is that it is composed of a number of stages that are executed sequentially, with a go/no-go decision after each stage (e.g., Cooper, 2001; 2008). Usually, the first stage involves determining requirements (and feasibility), and it is followed by development of a design in the second stage. Subsequently, this design is executed in the third stage and tested in the fourth, after which the product is released in the final stage (and then sometimes maintained). Plan-driven methods thus thrive in a context characterized by relatively little uncertainty (although Johansson, 2014, describes how gates may play an important role in reducing ambiguity and uncertainty) and in which a project's requirements can be adequately assessed up front and specialized/dedicated teams may handle the various stages of the development process.

Over time, scholars further developed and customized the plan-driven approach so that it fitted various circumstances. For example, depending on a project's complexity, some stages can (or should) be omitted, overlapped, or repeated (Cooper, 2008). In this respect, a complex project is likely to benefit from additional development stages compared to a simple development project, which likely requires fewer stages (Cooper, 2008). Furthermore, in the case of overlapping stages, also referred to as "concurrent engineering," a stage may start before the previous one is completed, thereby potentially reducing overall development time (Clark and Fujimoto, 1991; Terwiesch et al., 2002). While this approach increases the risk of redundant work, as the downstream stage may apply upstream information that is potentially subject to change later on (Mitchell and Nault, 2007), successful implementations of concurrent plan-driven approaches have been found in, for example, the automotive industry (Pechmann et al., 2015) and telecom industry (Lin et al. 2008).[2]

[2] Previous research advises against concurrency in cases in which the downstream stage is (highly) sensitive to upstream changes, because this circumstance may result in an endless problem-solving cycle (Cantamessa and Villa, 2000; Loch and Terwiesch, 2005).

Meanwhile, in the software industry, the need for development speed in combination with the desire to limit problem-solving cycles between upstream and downstream stages resulted in the Agile development method (Paluch et al., 2020; Annosi et al., 2020; Gonzalez, 2014; Tura et al., 2017). In contrast to the plan-based method, this method assumes high levels of uncertainty, because a project's requirements—often referred to as "features"—cannot be reliably determined up front. The approach is characterized by many short development iterations (Conboy, 2009; Dingsøyr et al., 2010), executed by a dedicated team to facilitate communication. Features are typically added or adjusted until time runs out. As such, Agile development also requires substantial customer feedback on the solution offered by the developer (Fowler and Highsmith, 2001). Features that need development are placed in a backlog that serves as input for the next iteration—a so-called "sprint" (Dingsøyr et al., 2010; Wood et al., 2013). This way of working implies that a software team can only begin a new task when the current task is completed, reviewed, tested, and demonstrated to be fully functional and free of bugs, and it prevents the generation of unexpected rework in later stages of the project owing to "almost finished" tasks of the kind that typically endanger a whole project.

Table 1 lists key differences in the assumptions and characteristics underlying the plan-driven and Agile development approaches. Whereas the Agile method aims to minimize the effects of changes at any time in the product life cycle, allowing for flexible/evolving requirements, the plan-driven method aims to minimize requirement changes (Karlström and Runeson, 2005; Bianchi et al., 2020; Paluch et al., 2020). In sum, time is the fixed variable within an Agile development context, and requirements are allowed to vary, whereas functionality is the fixed variable in a traditional development context.

Table 1. Fundamental assumptions and characteristics for plan-driven and Agile development.

	Plan-driven development	Agile development
Typical use:	Physical product development	Digital product development
Assumed uncertainty:	Low	High
Specifications:	Determined up front; specifications are fixed	Determined after each iteration; development time is fixed
Development process:	Linear development process	Iterative development process
Team composition:	Phase specific (benefits specialization)	Integrated team (benefits communication)
Customer involvement:	Typically low	High (regular feedback required)

Over time, both practitioners and scholars became interested in developing Agile-plan-based hybrids (Cooper and Sommer, 2016; Port and Bui, 2009; Edwards et al., 2019) such as the Agile–Scrum-Gate model (Cooper et al., 2019). In practice, under this model each stage of the project uses sprints, or iterations, that are time boxed (Cooper et al., 2019). Cooper and Sommer (2018: 19) explain: "an Agile–Stage-Gate hybrid embeds the Agile way of working within Stage-Gate stages […], replacing traditional project management tools and approaches, such as Gantt charts, milestones, and critical path planning, with Agile tools and processes." Important differences between the Agile–Stage-Gate hybrid and the traditional Stage-Gate approach include much more variable and tentative gate deliverables as well as leaner deliverables (e.g., fewer and shorter templates) (Cooper et al., 2019). Studies are increasingly reporting the effectiveness of such hybrid models. For instance, Karlström and Runeson (2005; 2006) report that both methods can effectively be combined in a single project. Cooper and Sommer (2016) also conclude that the two development methods can be compatible, even symbiotic, while stating that more research is needed to uncover the advantages, disadvantages, and challenges. Furthermore, Port and Bui (2009) conducted simulation experiments to conclude that a mixed development strategy outperforms a single development method in some cases. Cooper and Sommer (2018) study manufacturing firms experimenting with Agile–Stage-Gate hybrids and find that these companies benefit from increased design flexibility and improved productivity. They also find, in a similar vein to Žužek et al. (2020), that hurdles including managerial skepticism and dedicated resources must be overcome to reap those benefits. Finally, Edwards et al. (2019) conclude that the Agile–Stage-Gate approach is beneficial to larger manufacturing firms and SME manufacturers when it comes to

the overall success of new product development—even if only particular Agile project management practices are implemented (Žužek et al., 2020). In this respect, there is increasing support for the position that a hybrid development strategy may enhance, among other things, cost control, product functionality, team communication, and productivity. However, these studies report findings on development contexts in which a plan-based approach was "enhanced" with Agile elements—in a within-firm development context.

As development teams face increasing pressure to develop products better and faster, the need to collaborate with key stakeholders, inside as well as outside of the firm, has grown significantly (Pech, Heim, and Mallick, 2014). Interorganizational collaboration has long been recognized as an important antecedent of innovation (Faems et al., 2005; Hofman et al., 2017). Such collaboration may involve, for instance, suppliers, users, and customers and/or knowledge institutes (e.g., Belderbos et al., 2004; Van Echtelt, 2008; Brem et al., 2018), and it may contribute to both exploitative and explorative innovation (Belderbos et al., 2004). Important reasons to collaborate include, but are not limited to, access to complementary assets/knowledge, joint development of new resources, or the possibility to share (the sometimes very high) development costs (and associated risks) among the different participants (e.g., Faems et al., 2005). A collaborative development setting also involves specific challenges, including the mitigation of opportunistic behavior (Gulati, 1995; Alvarez and Barney, 2001), the avoidance of coordination gaps (Gerwin, 2004), or the role of contracts in collaborative new product development (Hofman et al., 2017).

We lack, however, an understanding of the dynamics that may arise in the context of collaborative system development, in which software and hardware are developed by different actors that maintain potentially conflicting development strategies (see Table 1). Although we know about various *functional* hybrid development strategies, these all pertain to a single-firm context. A collaborative setting in which software and hardware development must go hand-in-hand adds complexities that remain undiscussed in the current literature. In this respect, we do not know whether development hybrids (e.g., the Agile–Stage-Gate hybrid model) are (or are not) a solution for a *collaborative* development project in which the two conflicting development methods are maintained in their native form to develop an embedded system.

3. Research Method

To explore the implications and consequences of using conflicting development methods in a collaborative project, we conducted an in-depth case study (Yin, 2009) on a collaborative embedded systems development project in the automotive industry. We adopted a process research approach (Langley, 1999; Langley et al., 2013; Van de Ven et al., 2000) and subsequently drew on systems thinking (Sterman, 2000) to uncover the project dynamics that emerged over time. This research design is particularly appropriate given our investigation's exploratory and temporal nature, which renders cross-sectional variance studies less suitable.

3.1. *Case Selection*

We selected a development project: 1) that constituted a collaborative embedded systems development project involving an Agile software development company that was contracted by a hardware manufacturer and that worked according to the plan-driven method; 2) that was part of a mature industry setting, to make sure the project was sufficiently representative of a typical embedded systems development project; 3) that had a sufficiently long duration (that of the selected project was well over 100 weeks) to allow us to observe patterns and relationships over time; 4) about which we were able to consult detailed information over the entire course of the project.

More specifically, the selected project involved a large and experienced software developer, SoftCo (which uses Agile, i.e., Scrum), contracted by an even larger manufacturer, ManuCo (which uses a concurrent plan-driven approach). SoftCo has a sound track record in developing highly customized and intricate software. For this particular project, SoftCo was hired by ManuCo to develop a custom solution.

Our primary unit of analysis for this research project is SoftCo. ManuCo was both a supplier to and customer of SoftCo. As a supplier, ManuCo provided the software team with the adequate testing setup. And as a customer, it specified the boundaries (e.g., through specifications and feedback) within which the software developer needed to operate. Indeed, in many development projects that include both software and hardware development, software development exists as a subproject in an environment composed of hardware development (Karlström and Runeson, 2006), making this a highly relevant setting.

By adhering to a plan-based approach—for instance, through setting strict up-front specification requirements and associated deadlines—ManuCo (unwittingly) enforced plan-based influences on SoftCo's Agile approach.

Furthermore, while ManuCo had a dual role (of both supplier and customer), the power distance between the two organizations implied SoftCo had to synchronize to ManuCo's schedule to a considerable extent. This power distance also meant that the immediate consequences of the interaction were especially visible on the software developer's side.

3.2. *Data Collection*

Our data consists of qualitative data in the form of progress meeting presentations; these were supplemented with interview data for clarification and triangulation (Yin, 2009). The main data consists of 83 SoftCo progress meeting presentations—these detail the progress of the project and emerging challenges or problems for the project team—as well as 81 meeting presentations for SoftCo's senior management team (which oversaw this strategically important project). These presentations were delivered over a period of 131 weeks. The weekly presentations included the most recent detailed information on topics such as: feature development milestones, development velocity and progress, key project activities (feature development and/or testing), requirements and specifications, hardware availability, quality issues, technical issues, and so forth. By mainly relying on weekly (detailed) presentations, our research design minimized retrospective bias (Golden, 1992). To obtain a complete view of the project's development, we gathered additional data through conducting 11 semistructured interviews with various project informants, including the software project leader, software developers, and the manufacturing project leader (at ManuCo) (Langley, 1999). Interviews lasted on average 60 minutes, during which interviewees were asked to elaborate on the main developments of the project over time, including project planning, (changes in) the development method, interaction with the other firm, and problems. All interviews were recorded and subsequently transcribed. When necessary, one of the authors went back to validate the findings.

3.3. *Data Analysis*

Data analysis took place in several subsequent steps. First, we constructed a general case narrative from the interview data and an initial analysis of the data from the progress meetings. This involved codifying a story that served as a preliminary step "aimed at preparing a chronology for subsequent analysis" (Langley, 1999: 695). In this respect, the narrative served as a first step in developing a detailed chronological account of the development project, pointing to key causal relationships and themes (Langley, 1999). Table 2 offers an overview of the critical events and their timing.

Table 2. A chronological account of key events.

Timing (week)	Key events
May 2010	Start of the project; the initial feature adherence plan was developed and agreed upon.
June/July 2010	First accounts of unrealistic deadlines; growing development backlog.
September 2010	Failure to meet first mid-term deadline (number of features too low).
March 2011	Workshop to discuss software validation plan. Lack of input from ManuCo implied that no agreed-upon validation plan became available.
March 2011	Decision to focus primarily on feature development, implying no bug fixing, to comply with contractual demands.
June 2011	Realization that solution is inadequate (number of bugs). Testing hardware remains unavailable.
June 2011	Decision to focus primarily on testing and bug fixing.
July 2011	Anticipated project deadline not met (solution is unreliable).
Late 2011–early 2012	Testing hardware becomes increasingly available.
February–March 2012	Accounts that ManuCo cannot keep up with the development speed of SoftCo.
June 2012	New project deadline not met (performance issues).
June 2012	Project received a no-go decision (ManuCo will not use solution).

Subsequently, we systematically analyzed the progress meeting presentations and management team presentations. Through doing so, we produced a detailed event list (database) of the project's weekly dynamics. Next, we coded the event list to uncover relevant themes or project activities during the course of the development project. The resulting codes, including definitions and example quotes, are presented in Table 3.

Table 3. Coding scheme.

Code	Description	Example quote from progress presentation
Development backlog	Development backlog compared to progress as scheduled in project proposal/agreement. Typically, mentions that indicate that the project is behind schedule.	*"Not enough front-end development progress to meet the week 42 deadline"* (week 13) *"Not all expected features included"* (week 57)
Quality issues	Quality issues, such as (high levels of) bugs and/or mentions of robustness/performance of the solution.	*"Large backlog of 10.1 defects will directly impact stability of 10.2. Rate of resolution too slow"* (week 6) *"Robustness of the software is not good"* (week 58)
Possibility to test (and fix) features	The (non)availability of testing equipment needed for testing the software solution on ManuCo's hardware.	*"Lack of validation materials (cars)"* (week 61) *"Test materials not available"* (week 65)
Lack of specifications from ManuCo	Unclear project specifications and/or integration specifications.	*"First software in product unclear"* (week 1) *"No integration plan"* (week 57)
Pressure on feature development	Pressure on SoftCo to comply with contractual obligations.	*"High pressure put on SoftCo to respect next ManuCo milestones in terms of content and deadlines"* (week 58)
Regeneration of issues	Bug proliferation process that results from undetected issues in software solution.	*"Number of open issues […] is increasing"* (week 50) *"More issues than expected"* (week 68)
Client (ManuCo) overload	Inability of ManuCo to provide timely feedback to SoftCo.	*"Nonavailability of up-to-date [hardware]" (week 100)* *"[Client] seems not to be able to handle complexity of [solution]'*

The coded event list and interview data pointed to various project phases, in which a particular project activity or theme dominated others. Notably, the conflict—and associated change in dominance—between the quantitative dimension of the development (i.e., "development backlog") and its qualitative dimension (i.e., "quality issues") stood out. Figure 1 denotes this interaction over time. More specifically, Figure 1 illustrates the number of mentions, depicted as an eight-week moving average, that were made of the development backlog (in black) and quality issues (in grey) during SoftCo's core team's weekly progress presentations. Using temporal bracketing (Van de Ven and Poole, 1995), we were able to divide the total stream of events into four episodes (A to D) to characterize further the (sequence of) key events and related themes that defined this development project. Note that the different episodes in Figure 1 do not coincide with standard (plan-driven) phases; instead, the episodes each mark a significant change in focus or activity within the software development project.

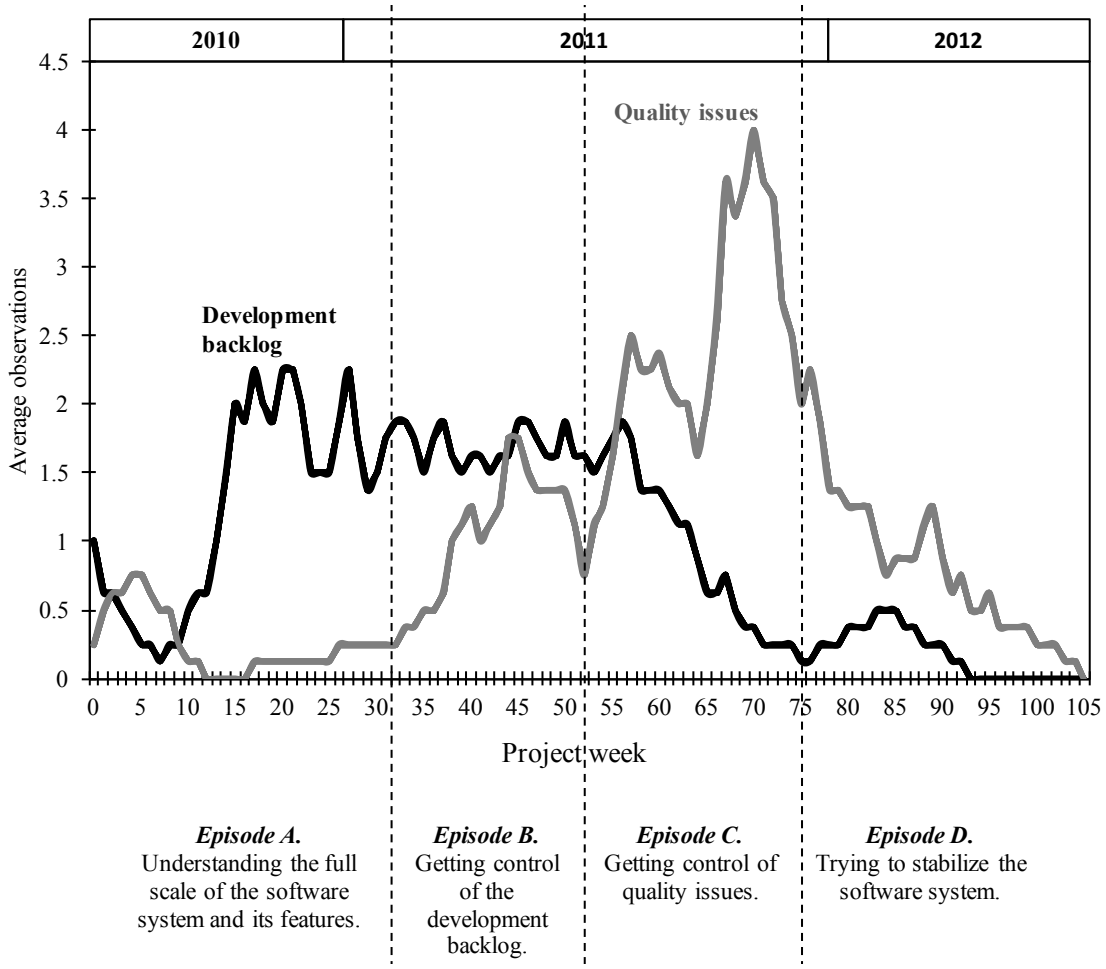

Fig. 1. Episodic phases underlying the development project, in terms of quantity versus quality of the solution.

In addition, we drew on systems thinking (Sterman, 2000) to identify the core mechanisms underlying the dynamics of these episodic phases and their causal connections. To better understand the observed processes, we developed a causal loop diagram (CLD). These are typical in system dynamics (Sterman, 2000) and increasingly common in management studies (e.g., Van Oorschot et al., 2013; Perlow et al., 2002). Individuals and organizations and the interactions between them constitute dynamic feedback systems that generate complex behavior (Lin et al., 2006; Metallo et al., 2021), and CLDs are a powerful tool for representing the core dynamics within such systems in terms of feedback (Lin et al., 2006). As such, the use of CLDs has a long tradition in academic research.

4. Findings

4.1. *Case Narrative*

The findings are presented as follows. First, we present the case narrative according to the identified temporal brackets. Figure 1 illustrates the four main brackets or episodes (A to D). As we have explained, this figure shows the evolution of SoftCo's focus on its two core activities, namely development (quantity, development backlog) versus testing (quality, bugs/issues). The associated narrative aims to clarify critical dynamics that were triggered by the collaborative development project and subsequently grounded the feedback loops in the CLD (see Figure 2). Using this causal loop diagram, we codified the potentially vicious keeping up appearances process.

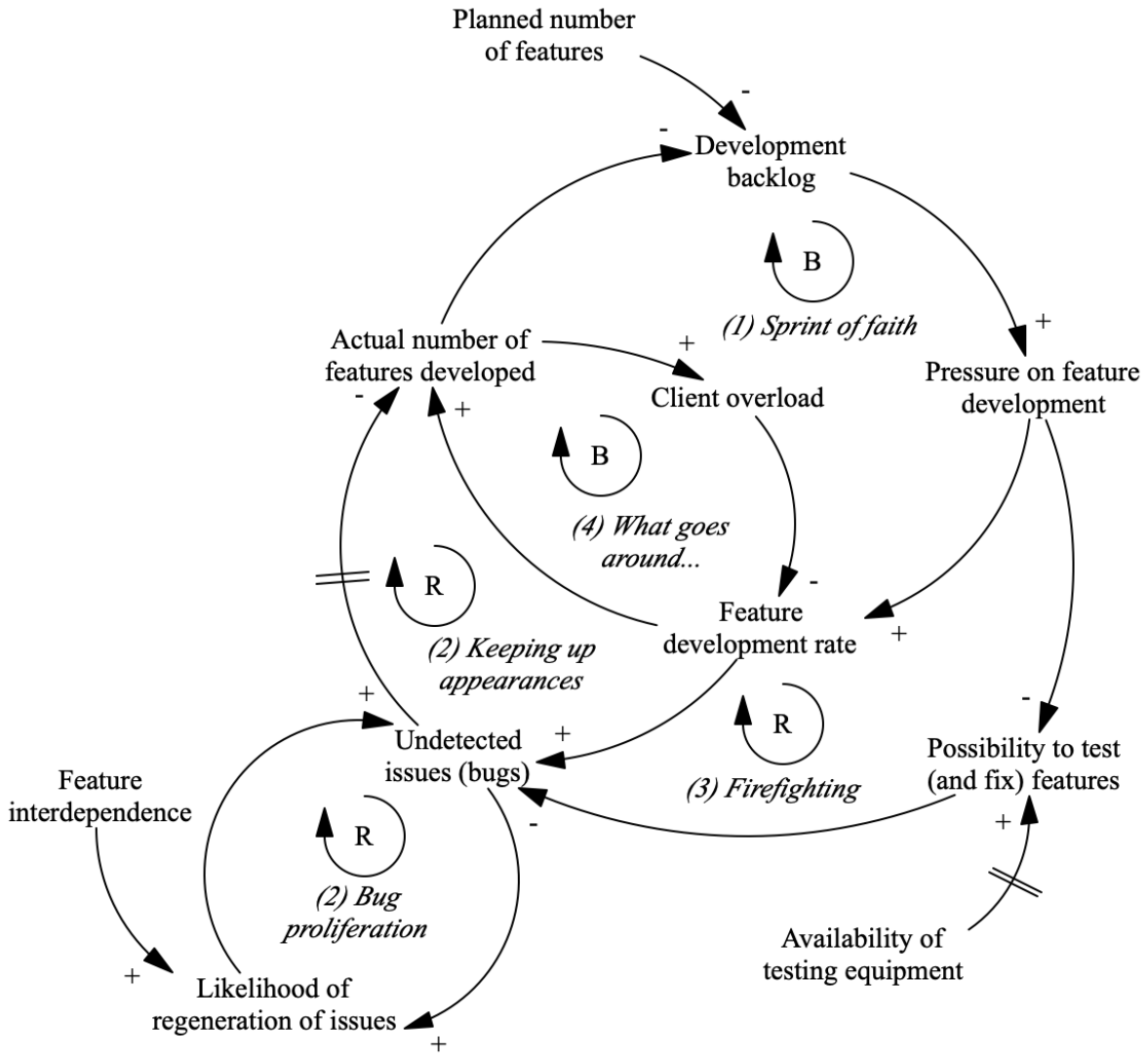

Fig. 2. Causal loop diagram of the keeping up appearances process. The labels B and R denote the nature of the feedback loop: balancing and reinforcing, respectively; the + and – denote the polarity of the causal link, while the // denotes a substantial delay (see Sterman, 2000).

4.1.1. *Episode A: A false start due to role conflict*

The project commenced in May 2010. An initial feature adherence plan was developed in the requirement definition phase. This plan stated, among other things, the number of features that needed to be developed as well as the deadlines for delivering those features. In this respect, the first 10 features were planned for September 2010, and feature number 150 (the final feature) was scheduled for July 2011. After agreeing on the number and nature of specifications as well as on their corresponding deadlines, the contracted software company and its client (the hardware company) started development. As tasks among the partners were divided in a modular fashion, the plan seemed to allow SoftCo's team to develop using an Agile approach without too much interference from ManuCo's plan-driven approach. More specifically, ManuCo attempted to create discrete subsystems that allowed independent teams to work on the project. As such, after the requirement definition phase, both organizations started to work on the design concurrently.

Nevertheless, developing in an Agile manner can only be truly effective if each sprint (or feature development cycle) consists of both development and testing (including customer feedback). Given the nature of this collaborative embedded systems development project, testing required the input of validation equipment (i.e., hardware) that ManuCo developed. Because ManuCo did not work with short iterative cycles (e.g., through rapid prototyping), but with one long design phase, SoftCo was forced to wait for this validation equipment to become available. Although some testing and bug fixing was possible (mainly through simulations), a truly thorough analysis of the developed features was not possible, which kept SoftCo in the dark about the actual quality of its

work. This situation was also acknowledged during an interview with the project manager of the software team. He reflects:

> *"Actually, the whole project started too early. Against our better judgment, we started the project, even though everyone knew the [testing] resources were not available."*

SoftCo's core team meeting notes also mentioned in this regard:

> *"Some impact on project milestones due to unreliable estimates and resource [i.e., hardware] availability"* (Core team meeting, July 2010).

As a result, the software team resorted to "sprints of faith" (Figure 2, "sprint of faith" loop). This implies that features were being developed without thorough testing in an attempt to keep the development speed in line with expectations and contractual obligations, thereby significantly increasing the length of their Agile iterations. This situation finds its origin in role conflict (Katz and Kahn, 1978), whose presence was perceived by the software team. Role conflict occurs when there are incompatible demands placed on an actor. That is, role conflict is experienced when "we find ourselves pulled in various directions as we try to respond to the many statuses we hold" (Macionis et al., 2010: 129). In other words, role conflict concerns the management of two (or more) pressures in a context in which compliance with one pressure will make it difficult to comply with the other pressure(s) (Katz and Kahn, 1978).

In this respect, the more the software company attempted to comply with the plan-driven demands of its client, the more it forwent its Agile, highly iterative development cycles (which are required for successful development). Early on in the project, this tension was fueled significantly by plan-based contractual agreements on specifications and associated deadlines. This situation caused a significant delay in the project quite soon after it had started—that is, a false start. The "feature development rate" was lower than initially planned owing to the many "undetected issues (bugs)," and this resulted in a rapidly increasing feature development backlog. As ManuCo's project manager observed:

> *"I remember the presentation from SoftCo stating the next deliveries would be a little delayed but after that we would make up some time and in June 2011 everything would be done. This was not […] the case in the end…"*

Notably, as the gap between planned and actually developed features kept increasing, so did the pressure on SoftCo to adhere to the (plan-driven) schedule. This situation triggered Episode B.

4.1.2. *Episode B: Keeping up appearances over bug proliferation*

The delay resulted in intense discussions and tension between SoftCo and ManuCo. The latter insisted on maintaining the project scope (150 features) and deadline (mid-2011). SoftCo replied that this was simply impossible due to the size and complexity of the desired solution. Nevertheless, in an attempt to respond to ManuCo's pressure and contractual demands, a period of *gear-up* was announced. During this episode, SoftCo mapped the backlog, and additional development resources were allocated. The emphasis at this stage was entirely on feature development (i.e., quantity over quality):

> *"Feature development until [April 2011], so no bug fixing"* (core team meeting, March 2011).

By directing all attention to feature development, the team was keeping up appearances (Figure 2, "keeping up appearances" loop) in an attempt to keep ManuCo satisfied and, as such, reduce tensions by "adhering to" contractual demands. Although the team knew that feature quality would likely turn out to be (too) low without thorough testing, the feature development rate would at least look good on paper. SoftCo reflects:

> *"A part of the problem was that many features were incomplete or not finished. They didn't work or only partly [worked]. Therefore, we decided to finish these features first, resulting in [a] full-feature, full-bug [solution]."*

Testing would only be done after all features were developed. However, development was increasingly hampered because there were already many problems with the software (i.e., bugs) and because (parts of) features were strongly interrelated. In other words, a vast number of bugs were incorporated into newly developed features, because the latter were based on features that had been developed earlier in the project and already suffered from low quality:

> *"The software behaves strangely in [the product]" (Core team meeting, June 2011).*
> *"The performance is unacceptable" (Core team meeting, June 2011).*

In other words, Episode A was bound to result in undetected issues (bugs), restraining the "actual number of features developed" and increasing the development backlog. This increased the "pressure on feature development," as the team had to meet plan-driven contractual agreements. As such, resources were increasingly directed toward feature development—this was an attempt to boost the feature development rate—at the cost of the "possibility to test (and fix) features." As resources constitute a zero-sum game, resources dedicated to development were not available for testing, and vice versa. While this full-feature, full-bug "strategy," initiated to keep up appearances, initially had a positive influence on the feature development rate and, therefore, on the actual number of features developed, this approach was subjected to strong policy resistance (Senge, 1990; Sterman, 2000). More specifically, the quantity of undetected issues (bugs) also dramatically increased as a result of a structural lack of bug testing/fixing. However, the feature development rate at this time hid this fact (Figure 2, "keeping up appearances" loop). Over time, the accumulation of software bugs had an increasingly negative influence on the actual number of features developed.

In developing representative tests to detect bugs, SoftCo remained dependent on ManuCo's validation equipment. More specifically, this equipment would allow SoftCo to test the software on the latest available subcomponents. Nevertheless, at this stage, such validation materials were still not a development priority for ManuCo, which was directing its focus toward other hardware involved in the development. This was also identified during one of the interviews with a SoftCo project manager:

> *"There was no one at ManuCo who was responsible for organizing all the validation equipment and bringing it up to date."*

Moreover, new software bugs were generated by old ones that remained undetected. This "bug proliferation" process is self-reinforcing: the more undetected bugs there are in a piece of software, the higher the probability that they will spread and generate even more bugs (Lin et al., 2008). This process is especially destructive if there is a strong interdependence between features. In our case study, the full-feature, full-bug approach, initiated by the incompatible development methods and fueled by role conflict (Episode A), drove an inclination on SoftCo's part to keep up appearances. This self-destructive strategy, intended to keep the client satisfied, initiated the vicious bug proliferation loop (Figure 2, "bug proliferation" loop). As a result, the actual number of features developed fell even further behind schedule. SoftCo was forced to keep on developing features as a result of its contractual agreements with ManuCo (specific features with associated deadlines) but was increasingly frustrated in executing this task due to the limited availability of validation resources.

Meanwhile, ManuCo did not yet have a validation plan describing how to precisely integrate the software into its hardware solution. Such plans should detail integration steps and typically also prescribe which components have priority over others, all of which would have been valuable input for SoftCo's development. One of SoftCo's managers mentioned during an interview:

> *"In March 2011, together with ManuCo, we organized a workshop to look at the validation plan and to decide which features needed to be delivered first to validate the system. This workshop was a total failure because [ManuCo] did not have a plan […]"*

4.1.3. *Episode C: Firefighting*

By the time the validation equipment and plan from ManuCo finally became available to SoftCo (following the plan-driven schedule of the hardware developer), the actual status of the project also became painfully clear. Bugs, which had remained hidden for an extended time, were discovered at a truly alarming rate. Bugs that were solved led to the discovery of yet more bugs. Testing became an extraordinarily tedious and inefficient ordeal, so much so that the decision was made to stop it completely:

> *"[At a certain point] I said to the team, stop with testing because we know there are 1,200 bugs. We first need to cut these back to 600… That will take us four weeks. After that you are allowed to test again."*

The software team was unmasked at this stage, as many (nested) problems within the software solution were exposed to the hardware developer. Whereas the software team was initially pressed to develop more features while having a decreased ability to test them, this next phase was characterized by an opposing strategy: full emphasis on bug testing and fixing (Figure 2, "firefighting" loop). This drove the "firefighting" loop aimed at combating the many undetected issues that resulted from the keeping up appearances and bug proliferation loops. This further delayed the collaborative project and resulted in Episode D.

More specifically, fixing issues to stabilize the system took almost 75% of the time it took to develop all features (39 weeks compared to 54 weeks), which was much longer than planned. As a result, despite all SoftCo's increased efforts (e.g., overtime and an increase in capacity), the deadline was severely missed.

4.1.4. *Episode D: What goes around…*

Although many bugs were fixed during the previous episode, the stability and performance of the overall solution left much to be desired. SoftCo therefore decided to give top priority to *stabilizing* the system in an attempt to finish the project before June 2012 (a full year later than the initially agreed completion date). To finish the project before the new deadline, SoftCo significantly prioritized the project, including by allocating additional resources to it. And with the bug proliferation loop under control, SoftCo was finally able to work in an Agile manner. As a result, the feature development rate increased substantially. Nevertheless, the improved workflow now strained its client, ManuCo, which also needed to step up its reviewing activities. ManuCo reflects:

> *"Receiving new software to validate every day was a nightmare for the team."*

As a result, the problems around the availability of validation equipment remained, though now they were occurring because ManuCo could not handle the increased inflow of work from SoftCo. ManuCo realized too late that the validation of the system would require a significant amount of work, and it had not considered such a workload in its schedule. This situation again delayed the feedback to SoftCo, once more frustrating development speed. Often, testing equipment was not available on the requested date, or hardware was equipped with the wrong specifications. SoftCo's core team meeting notes state, among other things:

> *"[Hardware is equipped with the] wrong software for testing" (core team meeting, February 2012)*
>
> *"Nonavailability of up-to-date test equipment" (core team meeting, March 2012).*

This "what goes around comes around" situation (Figure 2, "what goes around…" loop), is the culmination of the keeping up appearances process. As a whole, this process substantially diminishes the chances of success in collaborative development projects that try to combine Agile and plan-based development methods.

In the end, this situation caused further delays and disruptions, and the deadline was missed once more. SoftCo's solution received an overall no-go because its performance was insufficient. The missed deadline had significant negative financial implications for both firms. One of SoftCo's engineer's recalls:

> *"In [June] we received a no-go for the software part, which was a complete crisis for us."*

The complete causal feedback structure set out in Figure 2, which includes five main feedback loops and reflects a "shifting the burden" archetype (Senge, 1990), shows the dynamics that may arise due to the interaction of plan-based and Agile development methods in the context of collaborative system development. The causal loop diagram (Sterman, 2000), grounded in the case narrative and temporal brackets of the development project, identifies and explains the generative mechanisms and patterns underlying the observed development process.

5. Discussion and Implications

The complexity of today's embedded product development implies that software and hardware are often developed by different highly specialized firms that typically adhere to potentially conflicting development techniques. This study set out to investigate the dynamics underlying such collaborative embedded systems development. More specifically, we investigated a development context that is arguably becoming increasingly common, in which a plan-based hardware developer collaborates with an Agile software developer to develop complex embedded systems (Karlström and Runeson, 2006). Using an in-depth case study from the automotive industry, we observed the dynamics arising from the interaction between plan-based and Agile methods. Subsequently, by applying a process research approach that included event-based analysis, temporal bracketing, and systems thinking, we formalized a causal loop diagram (Figure 2) that captures the keeping up appearances process. Our generic theory, unfolding over four main periods, describes how tensions arise in such a collaborative development setting and how these tensions damage the project.

While conclusions drawn from a single case study require some caution, the findings presented in this paper provide important insights into the dynamics that underlie keeping up appearances. The process finds its origin in role conflict caused by mismatches between the development methods. Once initiated, a series of events unfolds along a self-destructive path. Notably, Episode B is key, as this period is characterized by self-reinforcing feedback. The negative influence of the keeping up appearances process grows exponentially during this period,

and therefore determines to a great extent the additional effort required to fix and finish the development project later on (if this is still possible at all).

Development approaches that combine plan-based and Agile methods are often referred to as "hybrid" development methods (Ghezzi and Cavallo, 2020; Paluch et al., 2020), or, more specifically, as Agile–Stage-Gate hybrids (Sommer at al., 2015; Edwards et al., 2019; Cooper et al., 2019). Examples of *functional* hybrid development approaches are becoming increasingly widespread (e.g., Cooper and Sommer, 2016; Karlström and Runeson, 2005, 2006; Edwards et al., 2019; Žužek et al., 2020). Some scholars suggest making a plan-driven approach more Agile by, for instance, introducing rapid prototyping (Cooper, 2008; Cooper and Sommer, 2016; Datar et al., 1997). Others propose making an Agile approach more plan driven by, for instance, changing the length of the iterative cycles or sprints (Port and Bui, 2009; Vigden and Wang, 2009). Yet other scholars advocate modularity, which allows different modules to be developed by utilizing different methods (Austin and Devin, 2009; Lenfle and Loch, 2010; Loch and Terwiesch, 2005). We contribute to the literature on Agile–Stage-Gate hybrids by studying a project that illustrated high interdependence between the different parts (implying a low possibility for modularity[3]). Furthermore, different highly specialized organizations working in collaboration developed software and hardware, maintaining conflicting development methods as they did so. We find that such a development context renders existing functional hybrid solutions unattainable (Kaisti et al., 2013). In this respect, our findings contribute to the literature by detailing a *dysfunctional* hybrid approach and serve to illustrate that plan-based and Agile approaches, in the context of collaborative software and hardware development, cannot readily be combined (cf. Cooper 2016; 2017; Edwards et al., 2019; Žužek et al., 2020). Our findings also imply that more research is required to uncover the boundary conditions for hybrid development tactics, notably in collaborative settings.

While product development success stories and best practices are widespread in the literature (e.g., Kahn et al., 2012), studies on *how* firms or projects fail are relatively rare (some notable examples are: Ring and Van de Ven, 1994; Tripsas and Gavetti, 2000; Van Oorschot et al., 2013; Walrave et al., 2011). Yet detailed knowledge of the reasons underlying failure might prevent managers from falling into similar traps. Our study contributes to this relatively small yet important body of knowledge by detailing a potentially vicious process by which a collaborative systems development project fails. That is, our theory serves to explain why a contracted software firm might deliberately follow a potentially self-destructive path through a series of seemingly rational decisions and actions.

Our theory responds to a long-standing call by Dougherty (1996), who argues that, despite a few notable examples (e.g., Lewis et al., 2002), scholars are failing to capture the tensions that underlie new product development adequately. Tensions have long been considered in organization and management science (e.g., Magnusson et al., 2009; Sheremata, 2000). In this respect, many studies highlight the need for managers to cope with conflicting and fluctuating demands (e.g., Dougherty, 1996; Lewis et al., 2002). Here, we focus specifically on tensions that may arise due to the interaction of plan-based and Agile development approaches in the context of a collaborative embedded systems development project.

5.1. *Managerial Contributions*

This study contains important insights for managers who are assigned to collaborative embedded systems development projects that involve a software developer and a hardware developer (cf. Tiberius et al., 2021). First of all, our in-depth case study illustrates in great detail how the project was effectively hindered by the combination of the two development methods favored by each party and the nature of the buyer-supplier relationship. This configuration resulted in a process that we call "keeping up appearances," in which the software company was forced to "adopt" a conflicting development strategy to manage the external pressure from its client. Interestingly, forcing an Agile supplier to comply with plan-driven demands makes the manufacturer (buyer) liable to sabotage its own development. In this respect, managers of such complex projects, informed by our results, should be able to better formulate a viable development strategy.

But what would such a strategy look like, given that the two methods seem to be fundamentally incompatible? One of the main advantages of the short cycles in Agile development is frequent testing. Although short cycles do not prevent software bugs or issues, these are discovered much earlier, and therefore the reinforcing issue-

[3] In this respect, our findings also serve to confirm our expectation that it is unlikely that modularization works for collaborative embedded systems development.

regeneration process (i.e., bug proliferation) is prevented (see Figure 2). Significant reductions of 40-90% in defect density are reported when teams perform frequent tests, as they do under Agile approaches (Nagappan et al., 2008). Our case analysis illustrates that SoftCo took about 39 weeks to try to fix all issues (Episode C and a part of Episode D) in an attempt to deliver a stable system to ManuCo. A 40% reduction of this time implies only 23 weeks would have been needed for testing. A 90% reduction implies only four weeks of testing. Such a reduction could have prevented the project from being unsuccessful. Of course, frequent testing is only possible when validation equipment is available. Indeed, the lack of such equipment in the early phases of the collaborative project was a major enabler of keeping up appearances (Figure 2). Interestingly, if SoftCo had delayed the start of its software development and waited for the validation equipment to become available, it could have maintained an Agile development approach. It is very likely that frequent testing would have subsequently reduced the time needed for fixing issues and stabilizing the system (Nagappan et al., 2008). This is depicted in Figure 3. In this respect, the Agile approach, initiated after a delayed start, is more likely to result in a fully operational system at the deadline, through a huge reduction of time and effort spent on fixing (regenerated) issues. Thus, the delayed start could have been used to wait for validation equipment to be developed. This potentially *functional* hybrid strategy allows both hardware and software teams to work with their own preferred method, thereby preventing the software team from keeping up appearances.

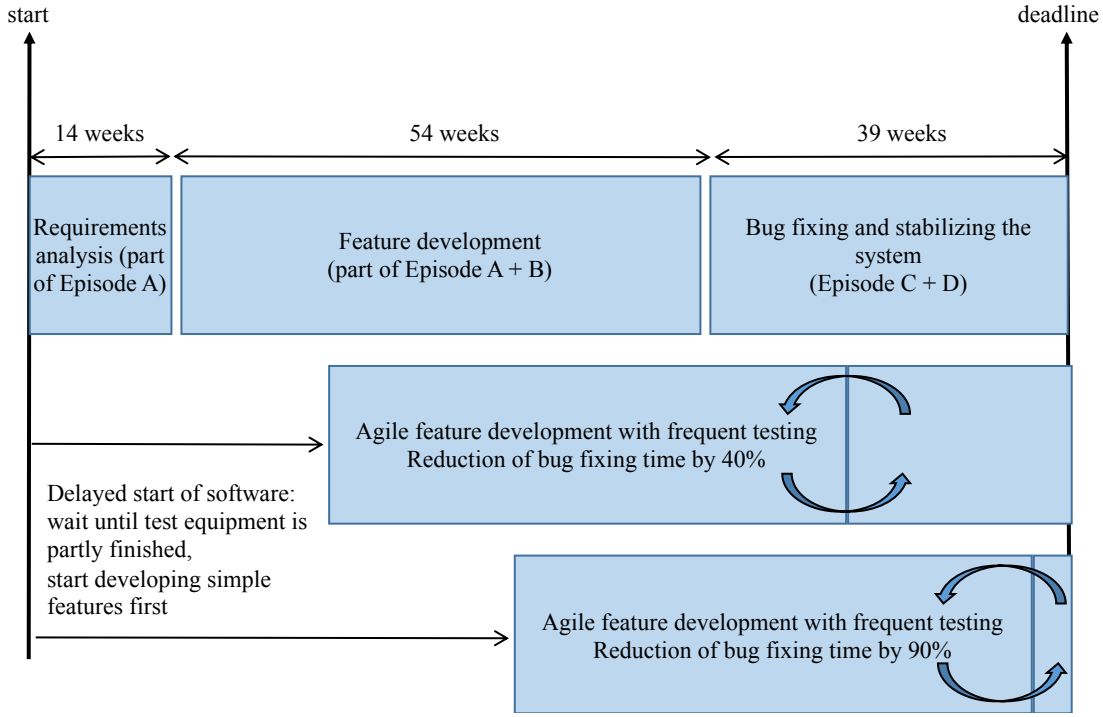

Fig. 3. Delayed start of the software team: *sequential Agile approach*.

As such, a sequential Agile approach in which the software team has a delayed start and waits for test equipment to be completed by the hardware team allows the software team to work in short cycles that include frequent testing, thereby preventing an escalation of issues. The time gained by avoiding these issues is not used to finish the project earlier but to delay the start of the software team's work. This approach differs from the existing hybrid strategies described in the literature thus far. First, it differs from the modularizing strategy as it allows for a close integration between software and hardware development, which is problematic under a modularization approach. Moreover, this approach, building on the Agile–Stage-Gate hybrid approach, could make the latter better applicable to collaborative settings. Of course, more research is needed to test such an idea for collaborative development in which both hardware and software teams are allowed to follow their own preferred development approaches.

6. Limitations and future work

Naturally, the fact that our findings are the product of a single in-depth case study has implications on the generalizability of the theory that we have developed (Dougherty, 1996). The process theory developed in this

paper was grounded in a particular collaboration. However, we postulate that our findings are relevant to complex development initiatives that entail a high level of interdependence and in which a plan-based hardware developer contracts an Agile software developer. Other forms of cooperation could potentially result in different processes for which further research is needed.

Here, we studied a development context that involved a plan-based hardware manufacturer. Cooper and Sommer (2016) and Edwards et al. (2019) argue that the plan-based approach (i.e., Stage-Gate) may be replaced by a hybrid Agile–Stage-Gate model. This implies that the plan-based approach is enriched with Agile elements. One might wonder whether the Agile–Stage-Gate model is, as such, more compatible with the native Agile method—perhaps even compatible enough to prevent the keeping up appearances process from happening. We leave this as an open question for future work.

Future work may also focus on the trust-versus-control aspects in collaborative embedded systems development projects (Bstieler, 2005). Our case seems to exhibit a lack of trust between the actors involved, as imposed contractual obligations were guiding—and limiting—the (software) development. Previous research has already established that trust is a central element in any relationship (Smets et al., 2013). However, the impact of trust policies relative to that of control policies is unknown in a collaborative *embedded* context. Perhaps the keeping up appearances process can be mitigated if formal controls (e.g., contracts) and informal controls (e.g., trust) are better balanced over time (see, e.g., Hofman et al., 2017; Smets et al., 2013).

7. Conclusion

This research set out to uncover the dynamics that may underlie complex collaborative embedded systems development projects in which conflicting development methods are maintained. We found that such projects may be subject to the keeping up appearances process, which is driven by role conflict and severely diminishes a project's chances of success. Building on the insights we developed, we propose the "sequential Agile approach," a development strategy that aims to counteract the vicious processes that our case study exposed. This potentially functional hybrid strategy allows the use of both plan-based and Agile approaches in one collaborative project, in which the Agile team delays the start of its part of the project until the plan-based team has made sufficient progress on its part.

References

1. Ågerfalk, P. J., Fitzgerald, B. and Slaughter, S. A. (2009). Flexible and distributed information systems development: State of the art and research challenges. *Information Systems Research,* **20**(3): 317–328.
2. Annosi, M.C., Martini, A., Brunetta, F. and Marchegiani, L. (2020). Learning in an agile setting: A multilevel research study on the evolution of organizational routines. *Journal of Business Research,* **110**: 554–566.
3. Austin, R. D. and Devin, L. (2009). Weighing the benefits and costs of flexibility in making software: Toward a contingency theory of the determinants of development process design. *Information Systems Research,* **20**(3): 462–477.
4. Baldwin, C. Y. and Clark, K. B. (2006). Modularity in the Design of Complex Engineering Systems. In A. Minai, D. Braha and B.Y. Yaneer (Eds.), *Complex Engineered Systems: Science Meets Technology*, 175–205. New York, NY: Springer.
5. Belderbos, R., Carree, M. and Lokshin, B. (2004). Cooperative R&D and firm performance. *Research Policy,* **33**(10): 1477–1492.
6. Bianchi, M., Marzi, G. and Guerini, M. (2020). Agile, Stage-Gate and their combination: Exploring how they relate to performance in software development. *Journal of Business Research*, **110**: 538–553.
7. Brem, A., Bilgram, V. and Gutstein, A. (2018). Involving lead users in innovation: A structured summary of research on the lead user method. *International Journal of Innovation and Technology Management*, **15**(03): 1850022.
8. Bstieler, L. (2005). Trust formation in collaborative New Product Development. *Journal of Product Innovation Management,* **23**(1): 56–72.
9. Cantamessa, M. and Villa, A. (2000). Product and process design effort allocation in concurrent engineering. *International Journal of Production Research,* **38**(14): 3131–3157.
10. Clark, K. B. and Fujimoto, T. (1991). Product development performance: Strategy, organization, and management in the world auto industry. Boston, MA: Harvard Business School Press.
11. Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research,* **20**(3): 329–354.
12. Cooper, R. G. (2001). Winning at new products: Accelerating the process from idea to launch. Cambridge, MA: Basic Books.
13. Cooper, R. G. (2008). The Stage-Gates idea-to-launch process—Update, what's new, and NexGen systems. *Journal of Product Innovation Management,* **25**(3): 213–232.

14. Cooper, R. G. and Sommer, A. F. (2016). The Agile–Stage-Gate Hybrid Model: A promising new approach and a new research opportunity. *Journal of Product Innovation Management,* **35**(5): 513–526.
15. Cooper, R. G. (2016). Agile-Stage-Gate hybrids: The next stage for product development. *Research-Technology Management*, **59**(1): 1–9.
16. Cooper, R. G. (2017). Idea-to-launch gating systems: Better, faster, and more agile. *Research-Technology Management,* **60**(1): 48–52.
17. Cooper, R. G. and Sommer, A.F. (2018). Agile–Stage-Gate for manufacturers – Changing the way new products are developed. *Research-Technology Management*, **61**(2): 17-26.
18. Cooper, R. G., Dreher, A. and Fürst, P. (2019). How Agile development works for manufacturers", Parts 1 & 2. *Center for Innovation Management Studies Management Report*, Part 1 in March-April 2019 & Part 2 in May-June 2019.
19. D'Adderio, L. and Pollock, N. (2014). Performing modularity: Competing rules, performative struggles and the effect of organizational theories on the organization. *Organization Studies,* **35**(12): 1813–1843.
20. Das, T. K. and Teng, B. S. (2000). A resource-based theory of strategic alliances. *Journal of Management,* **26**(1): 31–60.
21. Datar, S., Jordan, C., Kekre, S., Rajiv, S. and Srinivasan, K. (1997). New development structures and time-to-market. *Management Science,* **43**(4): 452–464.
22. Dingsøyr, T., Dyba, T. and Moe, N. B. (2010). *Agile software development, current research and future directions*. Berlin, Germany: Springer-Verlag.
23. Dougherty, D. (1996). Organizing for innovation. In S.R. Clegg, C. Hardy, and W.R. Nord (Eds.), *Handbook of organization studies*, 424–439. Thousand Oaks, CA: Sage.
24. Edwards, K., Cooper, R. G., Vedsmand, T. and Nardelli, G. (2019). Evaluating the Agile-Stage-Gate hybrid model: Experiences from three SME manufacturing firms. *International Journal of Innovation and Technology Management*, **16**(08): 1950048.
25. Faems, D., Van Looy, B. and Debackere, K. (2005). Interorganizational collaboration and innovation: Toward a portfolio approach. *Journal of Product Innovation Management*, **22**(3): 238–250.
26. Fowler, M. and Highsmith, J. (2001). The agile manifesto. *Software Development,* **9**: 28–35.
27. Ghezzi, A. and Cavallo, A. (2020). Agile Business Model Innovation in Digital Entrepreneurship: Lean Startup Approaches. *Journal of Business Research*, **110**: 519–537.
28. Golden, B.R. (1992). The past is the past—or is it? The use of retrospective accounts as indicators of past strategy. *Academy of Management Journal,* **35**(4): 848–860.
29. Gonzalez, W. (2014). Applying agile project management to predevelopment stages of innovation. *International Journal of Innovation and Technology Management*, **11**(04): 1450020.
30. Hofman, E., Faems, D. and Schleimer, C. (2017). Governing collaborative New Product Development: Towards a configurational perspective on the role of contracts. *Journal of Product Innovation Management,* **34**(6): 739–756.
31. Johansson, C. (2014). Managing uncertainty and ambiguity in gates: Decision making in aerospace product development. *International Journal of Innovation and Technology Management*, **11**(02): 1450012.
32. Kahn, B. K., Barczak, G., Nicholas, J., Ledwith, A. and Perks, H. (2012). An examination of New Product Development best practice. *Journal of Product Innovation Management,* **29**(2): 180–192.
33. Kaisti, M., Rantala, V., Mujunen, T., Hyrynsalmi, S., Könnölä, K., Mäkilä, T. and Lehtonen, T. (2013). Agile methods for embedded systems development: A literature review and a mapping study. *EURASIP Journal on Embedded Systems,* **15**: 1–16.
34. Karlström, D. and Runeson, P. (2005). Combining agile methods with stage-gate project management. *IEEE Software,* **22**(3): 43–49.
35. Karlström, D. and Runeson, P. (2006). Integrating agile software development into stage-gate managed product development. *Empirical Software Engineering,* **11**(2): 203–225.
36. Katz, D. and Kahn, R. L. (1978). *The social psychology of organizations*. Oxford, UK: Wiley.
37. Kortelainen, S. and Lättilä. (2013). Hybrid modelling approach to competitiveness through fast strategy. *International Journal of Innovation and Technology Management*. **10**(5): 1340016.
38. Langley, A. (1999). Strategies for theorizing from process data. *Academy of Management Review,* **24**(4): 691–710.
39. Langley, A., Smallman, C., Tsoukas, H. and Van de Ven, A. H. (2013). Process studies of change in organization and management: Unveiling temporality, activity, and flow. *Academy of Management Journal,* **56**(1): 1–13.
40. Lenfle, S. and Loch, C. (2010). Lost Roots: How project management came to emphasize control over flexibility and novelty. *California Management Review,* **53**(1): 32–55.
41. Lewis, M. W., Welsh, M. A., Dehler, G. E. and Green, S. G. (2002). Product development tensions: Exploring contrasting styles of project management. *Academy of Management Journal,* **45**(3): 546–564.
42. Lin, C. H., Tung, C. M. and Huang, C. T. (2006). Elucidating the industrial cluster effect from a system dynamics perspective. *Technovation,* **26**(4): 473–482.
43. Lin, J., Chai, K. H., Wong, Y. S. and Brombacher, A. C. (2008). A dynamic model for managing overlapped iterative product development. *European Journal of Operational Research,* **185**(1): 378–392.
44. Loch, C. H. and Terwiesch, C. (2005). Rush and be wrong or wait and be late? A model of information in collaborative processes. *Production and Operation Management,* **14**(3): 331–343.
45. Macionis, J. J. and Gerber, L. M. (2010). *Sociology*. Toronto, ON: Pearson Canada.
46. Magnusson, M., Boccardelli, P. and Börjesson, S. (2009). Managing the efficiency-flexibility tension in innovation: Strategic and organizational aspects. *Creativity and Innovation Management,* **18**(1): 2–7.

47. Meredith, J. R. and Mantel, S. J. (2011). *Project management: A managerial approach.* Hoboken, NJ: Wiley.
48. Metallo, C., Agrifoglio, R., Briganti, P., Mercurio, L. and Ferrara, M. (2021). Entrepreneurial Behaviour and New Venture Creation: the Psychoanalytic Perspective. *Journal of Innovation & Knowledge*, **6**(1): 35–42.
49. Mitchell, V. L. and Nault, B. R. (2007). Cooperative planning, uncertainty, and managerial control in concurrent design. *Management Science,* **53**(3): 375–389.
50. Nagappan, N., Maximilien, E. M., Bhat, T. and Williams, L. (2008). Realizing quality improvement through test driven development: Results and experiences of four industrial teams. *Empirical Software Engineering,* **13**: 289–302.
51. Pechmann, F. von, Midler, C., Maniak, R. and Charue-Duboc, F. (2015). Managing systemic and disruptive innovation: Lessons from the Renault Zero Emission Initiative. *Industrial and Corporate Change,* **24**(3): 677–695.
52. Peng, D. X., Heim, G. R. and Mallick, D. N. (2014). Collaborative product development: The effect of project complexity on the use of information technology tools and new product development practices. *Production and Operation Management*, **23**(8): 1421–1438.
53. Perlow, L. A., Okhuysen, G. A. and Repenning, N. P. (2002). The speed trap: Exploring the relationship between decision making and temporal context. *Academy of Management Journal,* **45**(5): 931–955.
54. Port, D. and Bui, T. (2009). Simulating mixed agile and plan-based requirements prioritization strategies: Proof of concept and practical implications. *European Journal of Information Systems*, **18**(4): 317–331.
55. Paluch, S, Antons, D., Brettel, M., Hopp, C., Salge, T.-O., Piller, F. and Wentzel, D. (2020) Stage-gate and agile development in the digital age: Promises, perils, and boundary conditions. *Journal of Business Research,* **110**: 495–501,
56. Ring, P. S. and Van de Ven, A. H. (1994). Developmental processes of cooperative interorganizational relationships. *Academy of Management Review,* **19**(1): 90–118.
57. Senge, P. M. (1990). The fifth discipline: The art & practice of the learning organization. New York City, NY: Doubleday Business.
58. Sheremata, W. A. (2000). Centrifugal and centripetal forces in radical new product development under time pressure. *Academy of Management Review,* **25**(2): 389–408.
59. Smets, L. P. M., Van Oorschot, K. E. and Langerak, F. (2013). Don't trust trust: A dynamic approach to controlling supplier involvement in new product development. *Journal of Product Innovation Management,* **30**(6): 1145–1158.
60. Sommer, A. F., Hedegaard, C., Dukovska-Popovska, I. and Steger-Jensen, K. (2015). Improved product development performance through Agile/Stage-Gate hybrids. *Research-Technology Management,* **58**(1): 34–45.
61. Sterman, J. D. (2000). Business dynamics: Systems thinking and modeling for a complex world. New York City, NY: McGraw Hill.
62. Terwiesch, C., Loch, C. H. and De Meyer, A. (2002). Exchanging preliminary information in concurrent engineering: Alternative coordination strategies. *Organization Science,* **13**(4): 402–419.
63. Tiberius, V., Schwarzer, H. and Roig-Dobón, S. (2021). Radical innovations: Between established knowledge and future research opportunities. *Journal of Innovation & Knowledge*, **6**(3): 145–153.
64. Tura, N., Hannola, L. and Pynnönen, M. (2017). Agile methods for boosting the commercialization process of new technology. *International Journal of Innovation and Technology Management*, **14**(03): 1750013.
65. Tripsas, M. and Gavetti, G. (2000). Capabilities, cognition, and inertia: Evidence from digital imaging. *Strategic Management Journal,* **21**(10/11): 1147–1161.
66. Van de Ven, A. H. and Poole, M. S. (1995). Explaining development and change in organizations. *Academy of Management Review,* **20**(3): 510–540.
67. Van de Ven, A. H., Angle, H. L. and Poole, M. S. (2000). *Research on the management of innovation: The Minnesota studies*. Oxford, NY: Oxford University Press.
68. Van Echtelt, F. E. A., Wynstra, F., Van Weele, A. J. and Duysters, G. (2008). Managing supplier involvement in new product development: A multiple-case study. *Journal of Product Innovation Management,* **25**(2): 180–201.
69. Van Oorschot, K. E., Akkermans, H., Sengupta, K. and Van Wassenhove, L. N. (2013). Anatomy of a decision trap in complex new product development projects. *Academy of Management Journal,* **56**(1): 285–307.
70. Vatananan, R. S. and Gerdsri, N. (2012). The current state of technology roadmapping (TRM) research and practice. *International Journal of Innovation and Technology Management*, **9**(04): 1250032.
71. Vigden, R. and Wang, X. (2009). Coevolving systems and the organization of agile software development. *Information Systems Research,* **20**(3): 355–376.
72. Walrave, B., Van Oorschot, K. E. and Romme, A. G. L. (2011). Getting trapped in the suppression of exploration: A simulation model. *Journal of Management Studies,* **48**(8): 1727–1751.
73. Wood, S., Michaelides, G. and Thomson, C. (2013). Successful extreme programming: Fidelity to the methodology or good teamworking? *Information and Software Technology,* **55**(4): 660–672.
74. Yin, R.K. (2009). *Case study research: Design and methods*. Thousand Oaks, CA: Sage.
75. Žužek, T., Gosar, Ž., Kušar, J. and Berlec, T. (2020). Adopting agile project management practices in non-software SMEs: A case study of a Slovenian medium-sized manufacturing company. *Sustainability*, **12**(21): 9245.