Journal of the Association for Information Systems (2020) **21**(4), 864-900
doi: 10.17705/1jais.00623

RESEARCH ARTICLE

ISSN 1536-9323

How Much Method-in-Use Matters? A Case Study of Agile and Waterfall Software Projects and their Design Routine Variation

Babu Veeresh Thummadi[1], Kalle Lyytinen[2]
[1]University of Limerick, Ireland, veeresh.thummadi@lero.ie
[2]Case Western Reserve University, USA, kalle@case.edu

Abstract

Development methods are rarely followed to the letter, and, consequently, their effects are often in doubt. At the same time, information systems scholars know little about the extent to which a given method truly influences software design and its outcomes. In this paper, we approach this gap by adopting a routine lens and using a novel methodological approach. Theoretically, we treat methods as (organizational) ostensive routine specifications and deploy routine construct as a feasible unit of analysis to analyze the effects of a method on actual, "performed" design routines. We formulated a research framework that identifies method, situation fitness, agency, and random noise as main sources of software design routine variation. Empirically, we applied the framework to examine the extent to which waterfall and agile methods induce variation in software design routines. We trace-enacted design activities in three software projects in a large IT organization that followed an object-oriented waterfall method and three software projects that followed an agile method and then analyzed these traces using a mixed-methods approach involving gene sequencing methods, Markov models, and qualitative content analysis. Our analysis shows that, in both cases, method-induced variation using agile and waterfall methods accounts for about 40% of all activities, while the remaining 60% can be explained by a designer's personal habits, the project's fitness conditions, and environmental noise. Generally, the effect of method on software design activities is smaller than assumed and the impact of designer and project conditions on software processes and outcomes should thus not be understated.

Keywords: Software Development, Agile, Waterfall Methodology, Method-In-Use, Routine Variation, Method Fit, Mixed Methods, Silhouette Clustering

Sandeep Purao was the accepting senior editor. This research article was submitted on May 18, 2016 and underwent one revision.

1 Introduction

Some routines show a lot of variation; others do not. Some are flexible; others are not. Some are easy to transfer; others are not. These variations may seem like noise or bad measurement, but they are not. They are indications of underlying phenomena and dynamics. By unpacking routines, we can begin to apply ideas and theories from all branches of social and behavioral sciences to explain these kinds of differences (Pentland & Feldman, 2005, p. 794).

A long-established design science tradition in information systems (IS) has examined the creation and implementation of system or software development methodologies, or methods, and the impact of their use (Hirschheim, Klein, & Lyytinen, 1995; Russo, Wynekoop, & Walz, 1995). A primary premise in this scholarship is that we can and need to differentiate between the effect of using or not using a method—that is, the use of the method matters for software and its process quality and/or user satisfaction. Over the years, the main motivation of software organizations has been to adopt, design, and invest in a variety of methods ranging from early waterfall methods to recent agile variants (Gordon & Bieman, 1993). However, the actual effects of such choices and investments remain largely anecdotal (Lindvall et al., 2002), although some evidence suggests that, for example, agile methods result in faster development processes, higher user satisfaction, and improved system quality. Even here, results remain mixed (Berente & Lyytinen, 2007).

The development activity conditions under which a development method is used to an effect that changes the process and outcomes continue to be poorly understood. For example, when an agile method is chosen, the extent to which activities are affected remains unclear. The few detailed analyses of actual uses of methods such as the agile method in specific design contexts (e.g., Vidgen & Wang, 2009) are highly illuminating in that they report in detail agile design practices and articulate how the method shapes such practices. However, most relevant studies are largely descriptive and fail to capture how the method truly shapes specific aspects of the design activity.

One primary deficiency in prior studies is that most analyses assume that the whole method and its use constitute the unit of analysis. Consequently, these studies simply detect the presence or absence of a method in a particular setting. Such high-level treatment hides how the method actually shapes design practice. Abrahamsson et al. (2002) eloquently summarize the current state of the understanding: "Despite the high interest in the subject, no clear agreement has been achieved on how to distinguish agile software development from more traditional approaches. The boundaries—if such exist—have thus not been clearly established" (Abrahamsson et al. 2002, p. 8).

One way to overcome this conundrum is to be more diligent in choosing and theorizing about the unit of analysis when analyzing method use and its impact, particularly when the true effects of a given method on a designer's behavior can be readily identified. One promising approach to such analysis is adopting a routine lens and analyzing designers' behavior as an enactment of routines that are shaped by a method (Feldman & Pentland, 2003). Accordingly, a design routine can be defined as a set of (sequential and/or parallel) activities that are repetitively carried out by a designer to transform "some representational inputs into a set of material and representational outputs, leading ultimately to a generation of a design artifact that offers a set of functions for a community of users" (Gaskin et al., 2012, p. 2). According to this approach, a design method is viewed as an "ostensive" specification of a design routine that provides a prescriptive recipe (resource) for a designer's design activities to be performed, whereas a "performative" design routine is a designer's enactment of the routine of specific design activities carried out while following a given recipe (Glaser, 2017). A design method par excellence conveys an official and formal statement of the ostensive specification of a design routine, whereas a designer's situated design practices when the method is "enacted" constitute the performative dimension of the design routine. In other words, the ostensive part captures how the organization formally and officially defines and expects its software design activities to unfold—that is, how the organization collectively "thinks" about its software process and accounts for its meaning and goals. The performative dimension captures the situated and embodied way in which designers in organizational settings carry out design activities that more or less comply with the method. Naturally, the ostensive enactment never fully captures the performative enactment (Feldman & Pentland, 2003). This has also been demonstrated in several method use studies in which the method "use" has been found to be adaptive and improvised (Russo et al., 1995; Feldman & Pentland, 2003), calling for constant "situational method adaptation" (Smolander, Tahvanainen, & Lyytinen, 1990).

Recently, some IS scholars have used a routine lens to observe variations in general design behaviors and to compare and explain design behaviors and their outcomes (Gaskin et al., 2014; Lindberg et al., 2016). However, they have not more deeply examined the extent to which the ostensive dimension of a method (Fitzgerald, 2000; Vidgen & Wang, 2009) serves as a true source of the detected design routine variation, defined here as the space of possibilities in the design routine's activity composition and order (Gaskin et al., 2012). Further, there is insufficient understanding of the extent to which different methods induce different levels of design routine variation—that is, whether different methods exercise differential effects on actual design routine variation and under what conditions. Overall, we know surprisingly little about how design methods are enacted in different settings, whether and how design activities are shaped by a given method, and whether the design outcomes truly differ because of the use of the method or because of some other factor or combination of factors (Berente & Lyytinen, 2007). To address this gap, we study the following questions:

RQ1: To what extent do design methods affect design routine variation during software development?

RQ2: Does such design routine variation differ across different methods, such as agile and waterfall methods, and in what ways?

We address these questions by formulating a design routine analysis framework that identifies four main sources of design routine variation during software development; we also discuss their theoretical foundations. This addresses RQ1 in that the framework offers an analytical, systematic approach for identifying and evaluating the impact of method use on design activities and software processes. To address RQ2, we empirically observe the extent to which the chosen method—in our case agile or waterfall—influences the performative dimension of a routine and the manifested differences in observed design routine variation. In particular, we probe the proportions of the observed design routine variation that are influenced by the followed method through a multicase study (Yin, 2017). The study focuses on the use of agile and waterfall methods and their impact on design routine variation in six midsize software projects over four years in a large software development unit responsible for managing bill of material (BOM) applications at a global original equipment manufacturing (OEM) automotive firm, referred to here as "Beta." The case study uses a mixed-methods research design and combines qualitative content analysis with computational techniques (such as Markov chain analysis, sequence analysis, and cluster analysis) to detect and explain structural variations in design activity composition and order.

We advance our argument as follows. In the next section, we review the extant literature on design methods, method use, and ostensive and performative dimensions of routines. Then, we review studies on method use and examine what we know about the effects of method use on design processes and outcomes. We briefly report our research methods and data collection and analysis techniques, followed by a section reporting the main research findings. We conclude by discussing the novelty of the introduced theoretical and analysis approach and evaluate how it can shape future studies on the use of software methods.

2 Theoretical Background

2.1 System Design Methods as Bundles of Ostensive Routines

Software development is widely recognized as a complex undertaking with many elements unaccounted for and often negative variation in its processes and outcomes. Because of this, software organizations have, for some time, paid attention to directing and reducing such variation to improve design processes and outcomes. One approach has been to prescribe ex ante specific ways of carrying out design activities expressed in a method intended to reduce the variation that follows (Glass, 1991). Since most methods share a reductionist worldview, it is widely assumed that better design solutions can be reached by following a prescribed set of sequential steps (Baskerville, Travis, & Truex, 1992; Fitzgerald, 1996). Since the mid-1960s, design methods have been invented, introduced, and applied to shape organizational responses to a large set of design tasks to avoid or mitigate the likelihood of a design or system failure that may negatively affect design quality, cost, or time parameters (Fitzgerald, 1996; Sommerville, 1996; Fitzgerald, 2000). Although most methods carry the ethos of control and seek reduction in design routine variation, significant differences in the proposed methods prevail because of differences in underlying philosophies, beliefs, and values or because of "product differentiation, personal ego, and territorial imperatives" (Fitzgerald, 1996, p. 11). Moreover, as Baskerville et al. (1992) posit, most methods are intended for large-scale development tasks that involve significant development time (Baskerville et al., 1992; Feller & Fitzgerald, 2000; Fitzgerald, 2000). However, these goals are not universally shared and additional factors (including organizational competencies and learning) have been recognized as reasons for choosing a specific method (Lyytinen, 1987). Accordingly, some methods like waterfall approaches may not be suitable for all situations because they may contradict the assumptions underlying the method (Baskerville et al., 1992; Petersen, Wohlin, & Baca, 2009). Generally, given the complexity of design processes and situations, no method is perfect. Even a light desk evaluation can easily find weaknesses in most methods for certain situations. Hence, one size does not fit all and organizations have thus learned to be more attentive in flexibly selecting a method for a given design situation (Laplante & Neill, 2004).

Due to the potential incompleteness of any one method, each method has been shown to leave designers significant degrees of freedom regarding use for a particular task (Berente & Lyytinen, 2007). Consequently, no daily set of design activities can ever faithfully reflect a given method. Examining the extent to which a designer has followed a method's prescriptions when evaluating design performance affords a limited understanding of the outcomes, because a significant amount of activity is always left unaccounted for. Indeed, research shows that significant deviations prevail at both contextual and individual levels regarding a method's enactment and the amount of design routine variation a designer can acceptably employ while still demonstrating coherence with any particular method (Russo et al., 1995).

Recent research on the dual nature of routines provides a fruitful lens for understanding and accounting for the observed gap between formally enforced and actually observed variation in the use of a method (Feldman & Pentland, 2003). Traditionally, changes in software development have been viewed as resulting from failures to adopt a given design method (Baskerville et al., 1992). However, Feldman and Pentland’s work offers an alternative explanation: design routine variation is an inherent property of method use and results from endogenous interactions between ostensive (structure) and performative (agency) aspects of a routine (Feldman & Pentland, 2003). Here, the ostensive dimension represents the ideal, abstract dimension of a routine, whereas the performative dimension is formed by intermittent and ephemeral instantiations of action employed while the actor is paying attention to ostensive “principles” (Latour, 1986). The ostensive dimension is “the ideal or schematic form of a routine” (Feldman & Pentland, 2003, p. 101), whereas the performative dimension “consists of specific actions, by specific people, in specific places and times. It is the routine in practice” (Feldman & Pentland, 2003, p. 101). The ostensive dimension can be thought of as a foundational “structure” that guides and orients the designer’s actions, while the performative dimension manifests in real, observed, design “actions.” Ultimately, the performative dimension contributes to the continued creation, maintenance, and modification of the ostensive dimension and, together, they form a “duality” (Giddens, 1984; Feldman & Pentland, 2003).

According to this view, the software development process is a bundle of performative routines carried out by designers and other stakeholders. The ostensive dimension of a software process consists of abstract ideas: some are embedded in general design methods, while others represent local or individual ideas and abstractions. Over time, such development ideas and abstractions are codified in written methods based on the repetitive execution of completing specific design tasks (learning by doing). Sometimes they are crafted from outside by absorbing design guidelines and ideas that have worked elsewhere (often those introduced by consultants or academics).

Scholars, however, often misconstrue a method as a single ostensive system that constitutes a monolithic object to be followed mechanistically (Feldman & Pentland, 2003). In reality, software development is continually socially constructed and, accordingly, local abstractions are influenced by multiple, often contradictory and incomplete extrasomatic sources. Designers continually attribute their “intersubjective interpretations” to design performances, which introduce irregularities into local design behaviors and related ideas (Lyytinen, 1986; Bijker, 1997). Design performances ultimately are like “improvised actions” that draw partially from the ostensive dimension and continually alter it by “situating” the ostensive dimension based on actors’ situated needs and opportunities (Suchman, 1987; Bourdieu, 1990). Thus, methods as ostensive specifications serve only as generic guides, standards, or principles and leave significant degrees of freedom for designers to modify, vary, and adapt the ostensive dimension and act differently (Berente & Lyytinen, 2007). According to the routine literature, varying degrees of freedom can be granted to the ostensive dimension, depending on the context—that is, depending on the setting, designers can enact different levels of choice surrounding the ostensive dimension in the performance of a routine. Yet each established and identified routine necessarily comes with a certain number of shared and understood expectations regarding the performance of the activity. To be and act like a method, each method must mitigate deviations and lower the degrees of freedom available to a designer to control design routine variation (Dionysiou & Tsoukas, 2013).

Overall, when we approach design methods as “bundles of ostensive routines” (Felin et al., 2012), we can sort software development into sets of separate design tasks and associated routines, such as gathering requirements. Each of these separate routines can be thought of as a family of activities. For some activities, designers use specific techniques—for example, collecting and recording requirements (use cases). In other activities, designers apply rules that specify the scope of included functions (e.g., house of quality) and follow protocols to validate the needs of users (e.g, quality reviews). Other activities use different techniques to estimate the cost of included functions (function analysis and cost-benefit evaluation) and so on. At the same time, the overall bundle of routines involved in gathering requirements comes with openness and ambiguity (which we refer to as “degrees of freedom”) in its ostensive specification. This allows for variations in the performative dimension of the routine, regarding, for example, the level of detail at which the use cases should be drawn, who should analyze them, or whether the scope of use cases should be determined at the start of the project) (Dionysiou & Tsoukas, 2013). Generally, we can say that any design method comes with an ostensive specification, but each design method can have varying effects on the performative dimension of the routine. This depends, among other things, on the uncertainty concerning the meaning of method specification and its usability in a given task or on the designer’s skills to apply it.

It has also been shown that designers strategize around their design performances and select strategically ostensive elements to which they adhere in order to demonstrate their accountability (Feldman & Pentland, 2003) or make the method fit their established habits and skills (Cohen & Bacdayan, 1994). Based on future interpretations, these choices are likely to change the ostensive element. However, according to Pentland (2003, p. 538), "there has been little attention to the issue of how to characterize these divergences." This prompts the question: *To what extent does the actual design performance conform to the specified ostensive element of a method and under what conditions does a designer deviate from it?* In the next section, we formulate a framework to identify and explain such design routine variation.

2.2 Design Routine Variation in Software Development

In the past, researchers have used surveys and related perceptual measures to detect variation in design routine (Pentland, 2003). Variation is generally couched in standardized question items that question, for example, respondents' perceptions of the maximum and minimum range of experienced variability in a task; variations are typically captured in the shifts and distances from the computed means. Unfortunately, such measures are not likely to capture the true variation because of anchoring effects, poor recall, and so forth. Such measures approach design routine variation as an aggregate across a whole project, which makes them inadequate for detecting the potential distance between the ostensive and performative dimensions (Vidgen & Wang, 2009). They fail to detect varying dimensions of this "unobserved design routine variation" because they capture variability in the activities' compositions, order, and perhaps interactions (Gaskin et al., 2012). Detecting true variations should help improve the understanding of how design activities differ across projects and under different contingencies, such as different methods used.

Recent studies that have focused on teasing out routine variation adopt the notion of a task, its related outcomes, and the concept of an activity as a pattern of actions that deliver those outcomes. The activity concept captures a fundamental idea that there are multiple ways that any given task can be done (Gaskin et al., 2012). One approach to capturing this variety involves separating between sequential variety (or order variation) and configural variety (or compositional variation). The latter refers to the variability across the activities, whereas the former captures the variation in the temporal structure of the activities (Gaskin et al., 2012). We use this line of thought to capture the latent, unobserved variation in design activities as a means to tap into the extent of design routine variation manifested both in the *composition* and the *order* of design activities (Roy, 1959; Feldman & Pentland, 2003; Hærem, Pentland, & Miller, 2015). The next question to explore is what determines the variation in both dimensions as the design process unfolds.

We identify and characterize four tentative sources that influence design routine variation—namely, method-induced variation, fitness-induced variation, agency-induced variation, and random variation. We deem these sources to be similar to sources of variance in classic measurement theory, i.e., analytically orthogonal and independent sources that nevertheless organically intertwine during any design activity (see Table 1). Some of these sources have been generally recognized in prior research (see Feldman & Pentland, 2003; Leonardi, 2011) and some studies have also recognized "attributes of the environment (fitness induced variation), individual cognitive processes, and the variety of an individual's experience (agency induced variation)" (Downey & Slocum, 1975, p. 765) as sources of routine variation. However, while the unique role of the ostensive dimension in inducing design routine variation has been generally recognized in past studies, it remains largely unaccounted for (see Feldman & Pentland, 2003; Leonardi, 2011). Most studies also recognize but do not extensively discuss random variation and its role.[1] Overall, the proposed framework is more complete than frameworks used in previous research because it adds method-induced variation and random variation as significant potential sources of design routine variation. Next, we discuss each variation type in more detail.

Method-induced variation: As noted, design methods serve the purpose of improving software processes because of their capacity to give systematic direction to development activity by controlling the range of variation in terms of how development activities are carried out (Fitzgerald, 2000, Pentland, 2003). Methods achieve this by conveying cognitive frames and establishing common ground for understanding and coordinating development; they also include normative principles (who should do what, when) that coordinate work dependencies. Finally, they impose standards for evaluating design decisions (such as rules of decomposition) (Lyytinen, 1986). When adopted and invested, methods act as primary (ostensive) sources for determining a range of design routine variations. Here, each design activity in such ostensive specification has specific outputs (task outcomes), which connect it to other activities and related routine bundles. Choices regarding how the connection is implemented influence the order of activities.

[1] This notion is similar to the idea of error term in ordinary least squares regressions—i.e., the remaining unaccounted variance.

For example, a waterfall method entails distinct sets of activities (such as design and coding) that are carried out sequentially (i.e., coding is not expected to start until design is finished) (Royce, 1970). In contrast, agile approaches seek to interlace design and coding activities by "layering" design outputs over time (see Figures 1 and 2; Abrahamsson et al., 2002). Activities complying with agile methods will shy away from documenting design activities as inputs to the next set of activities (Abrahamsson et al., 2002). Thus, selecting and adopting a method is likely to have a significant impact on how routines are composed and organized and on their variation.

Fitness-induced variation: Method developers cannot have perfect foresight of all the activities that need to be carried out during the development process. Designers must improvise and "retrofit" the method to unexpected organizational contingencies arising from method incompleteness or inadequacy (Glass, 1991; Kumar & Welke, 1992). Retrofitting increases the potential fit of the method (routine) with the situation (Levinthal & Rerup, 2006). Fitness-induced variations emanate either from initial omissions of specific, targeted guidelines or from the inappropriateness of the guidelines to the situation. Lack of fit can emerge from multiple sources and often comes as a surprise (Levinthal & Rerup, 2006). Contingency conditions include unexpected requirements, inappropriate technologies, and emerging insights that change the scope and functions of the system. As such, fitness variations are not detrimental—they add significant value to the design and are often necessary to render the final outcome functional.

Table 1. Design Activity Variation

Terms	Description
Variation/design routine variation	Variation highlights differences between two or more forms of the same activity. "Design routine variation" refers to multiple possibilities in the composition and order of activities and their variability across contexts. Researchers have used the construct to compare differences between two routines with similar outcomes (e.g., hiring routines; see Feldman & Pentland, 2003).
Method-induced variation	Differences in design practice due to the usage of a method(s), i.e., such activities would not be present in the same frequency if the ostensive element were not present.
Fitness-induced variation	Differences in design practice due to the structural contingencies in the design environment (e.g., unreliable technologies).
Agency-induced variation	Differences in design practices based on the habits or skills of the designer.
Random variation	Differences in design practice because of environmental noise such as fatigue or incidental misconception.

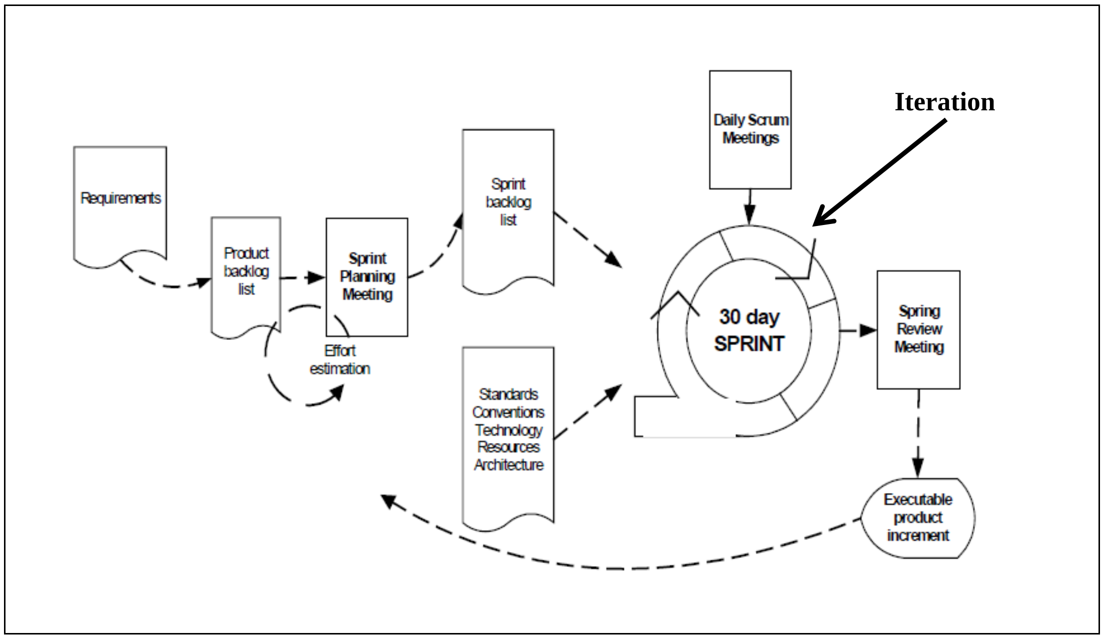


Figure 1. Iterations in the Agile Process (Abrahamsson et al., 2002, p. 28)

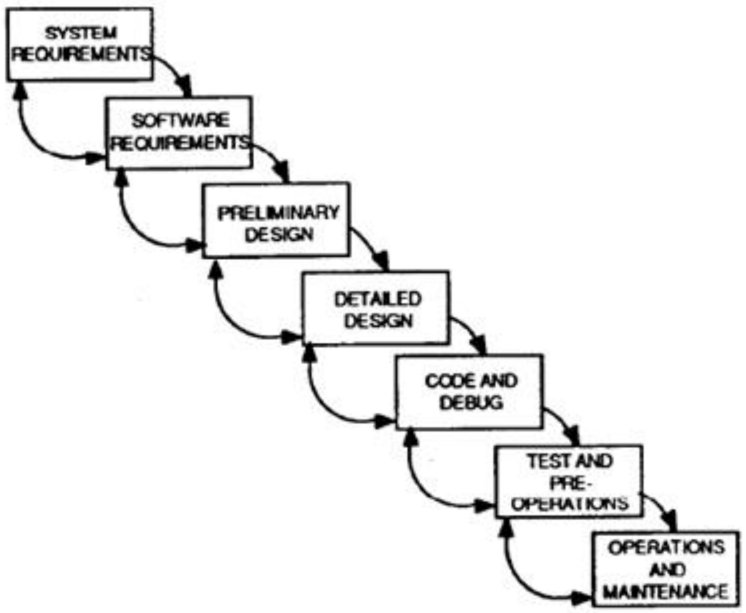


Figure 2. Iterations in Waterfall Methods (Royce, 1970, p. 330)

However, fitness variations do imply an additional range of (unexpected) routine variations, especially when contingencies start to dominate the design task. For example, a project manager in our field study observed that their actual process was far more iterative than planned due to unforeseen challenges in testing:

> *Yeah. When we are testing it, we found a few … issues and then they … did not match the requirements so we had to go through … iterations to make sure we implement those correctly. And … we found some different …, because when we had calculated … we had to make sure those were also implemented.*

Agency-induced variation: Agency ultimately determines development outcomes—in other words, the competency of actors who develop and work in the project matters. Accordingly, agency acts as a third source of design routine variation because individual skills, aptitudes, and experience vary. Designer behaviors rely mostly on experience and acquired knowledge, rather than on seeking fit with the provided method (Hirschheim, 2007). Differences in an actor's skills, experiences, and competencies modify performative routines (Cohen & Bacdayan, 1994, Feldman and Pentland 2003). Thus, agency-induced variation can have both negative and positive effects on design outcomes. If permitted to run unchallenged, developers will solely depend on their own, often variable skills and idiosyncratic views and ignore shared ostensive routines (such as strict documentation requirements, generating common test sets). This can make project-level coordination difficult or even impossible and initially motivates the introduction of methods when software projects begin to scale up. If allowed to run uncontrolled, reliance on personal skills can sow the seeds of large-scale failures. Nevertheless, individual skills and behaviors can significantly and positively influence software development outcomes and productivity (Scacchi, 2002). One actor (a developer) at Beta Corporation revealed how he found his skills to be insufficient for the project:

> *The beginning of the project. I was learning what is PD because I'm new to this whole thing. Because I am from HR group. I am used to dealing with the employee, human resources, and all these things. That's me basically. Even though I'm a JAVA developer, the domain knowledge I had was not that broad—the ins and outs of things. So, I had to learn. Then I learned that, and then I was given the introduction to what is LDM, what is this project.*

Random variation: Organizational behaviors always involve random noise. Due to the complexity of external and internal contingencies, behaviors in organizational settings are highly variable and random mutations in behaviors always emerge (Aldrich & Pfeffer, 1976). Random variation is often associated with unique, unexpected, singular conditions embedded in the environment, in an actor's psycho-physiological conditions (like fatigue), or in complex interactions in the interactive technological environment (Ciborra, Migliarese, & Romano, 1984).

In conclusion, the analysis of sources of design routine variation helps identify and clarify how performative routines are built and how they interrelate with ostensive approaches in complex ways (Dionysiou & Tsoukas, 2013). This helps frame our analysis that seeks to address our research questions: (1) *To what extent do methods influence design processes viewed as bundles of routines*? (2) *Do different methods influence performative routines differently?*

3 Research Design

We carried out a longitudinal four-year case study between 2010 and 2013 on software development in a large multinational manufacturing company (referred to as Beta), which is a large US manufacturing company known for its technological prowess in designing and manufacturing automotive vehicles. Our purpose was to identify sources of variation associated with methods in sampled projects and detect their effects on design routine variation. It was organized as a multilevel study, and it involved within- and cross-case analyses (Yin, 2017). The scale and complexity of the setting made the study an ideal setting for understanding sources of variation in routines. The research focused on the effects of enacting two contrasting software development methods—structured object-oriented method (waterfall) and agile methods.

3.1 Research Site

The study context was project teams that developed a large, critical family of applications within Beta called the Bill of Material foundation (BOM). This is a mission-critical suite of information systems that maintains critical part-related information associated with the design, manufacturing, supply chain, marketing, and service of cars. Over the past decade, Beta had been creating its BOM architecture in its information technology (IT) division and established a dedicated unit for this domain. The goal of the unit was to develop and maintain a family of applications that helped manage all product part-related information across the life cycle of a car. Due to the centrality of part information in anything that deals with designing, manufacturing, or selling cars, the unit was relatively large and viewed as highly important in the IT division and its projects were approached as mission critical. Each year, the unit ran multiple projects (large and small) to expand, revise, and improve part-related data management and service functions. During our study period, the unit underwent a major shift toward a new, more powerful data management platform. Related software development activities were globally distributed (in the United States, Europe, and India), because Beta runs geographically distributed design centers and has to share product and part information during design and marketing. Most of the final software was written in India in a development center owned and run by Beta. The design processes were digitally intense and used a suite of supporting software tools to share code and related information, such as test cases. This created a space to detect and analyze traces of design processes (Shoval & Isaacson, 2007).

In developing the BOM software, the unit relied on two methods for different projects. The first method was driven by the waterfall idea of proximal iterations, was based on object-oriented design and modeling, and used a local version of rational unified process.[2] The method had been developed and refined within the unit beginning in the mid-1990s. The second, a later approach, followed agile design and was based on Scrum (Schwaber, 1995). At the start of the study, Beta was mainly a waterfall practitioner; over the course of the study period, Beta significantly expanded its use of agile so that toward the end of the period, it mainly used agile design to develop applications.

3.2 Data Collection and Validation

We collected process data from six software projects with the goal of understanding the extent to which development routines differed in terms of the two design methods. Three software development projects were carried out with the object-oriented method and three followed agile methods. We collected data using replication logic from multiple projects to detect within and between variations in routines when a given method was used. This allowed us to locate overall variations within design processes and use related method data to infer the extent to which the variation was induced by the use of the method. The projects focused on developing specific features of the BOM database system and several front-end applications to manage or use product information during the car design process.

The projects were purposefully sampled to have comparable scale and complexity and were developed during roughly the same time period, using similar-sized teams. We sampled three Bill of Material (BOM) waterfall projects, referred to as BOM Search, PADB 1.4, and BOMFI in our data set (see Appendix A for more details about the project descriptions). The other three projects we sampled were agile projects (LCM 1.5-1.6, LCM 1.7, and LCM 1.8). We call these projects LCM projects because since most of them focused on managing product information and related change they were referred to as *lightweight change management* projects in the company. Agile methods were first used to develop this suite of applications for

[2] http://en.wikipedia.org/wiki/Rational_Unified_Process.

managing early part and product changes during car design—hence the name lightweight change. All projects shared the principal artifacts and critical infrastructural elements for software development covering project management, budgeting, personnel, and other support environments. Overall, the sampling offered the possibility of conducting a sort of quasi-experiment in that the projects were sampled to be similar and the main difference between them was the use of agile or waterfall methods (Shadish, Cook, & Campbell, 2002).

We chose to use semistructured interviews as our primary data collection method because they enabled us to systematically access fine-grained details of the design processes and outcomes. Toward this end, we developed a common interview protocol that focused on capturing the details of the design processes, their goals, actors involved, tools used, related inputs and outputs, key decision points, and so on (see Appendix B for the interview protocol). All interviews were conducted on-site, except in one case, where we interviewed an offshore team in India using phone and videoconferencing. During the interviews, we asked designers to show relevant system documentation, artifacts, or snippets of the actual implemented systems. We conducted 28 in-depth interviews with project managers and team members and validated the process models of their design processes using a thorough review process (see Table 2).

For each studied project, the data corpus was collected in two consecutive rounds. In the first round, we collected the primary process data, which were validated during the second round. In the first round of interviews (in 2010), we also interviewed the directors and vice presidents of the software development unit to understand the high-level and strategic reasons for following chosen methodologies. After this, we carried out interviews with software developers with different roles in BOM projects.

Table 2. Data Collection

Dates	Interview participants	Type of interview	Number of interviews
Jan-10	Vice president, three project managers	Face-to-face	3
Mar-10	Two project managers, business analysts	Face-to-face	3
Apr-10	Project managers, developers,	Face-to-face	5
May-10	Developers	Face-to-face	4
Dec-10	Project manager, developers	Face-to-face	2
Dec-11	Developers	Skype	2
Sep-12	Project managers, developers	Face-to-face	6
May-13	Project manager, developers	Face-to-face	3
Total			28

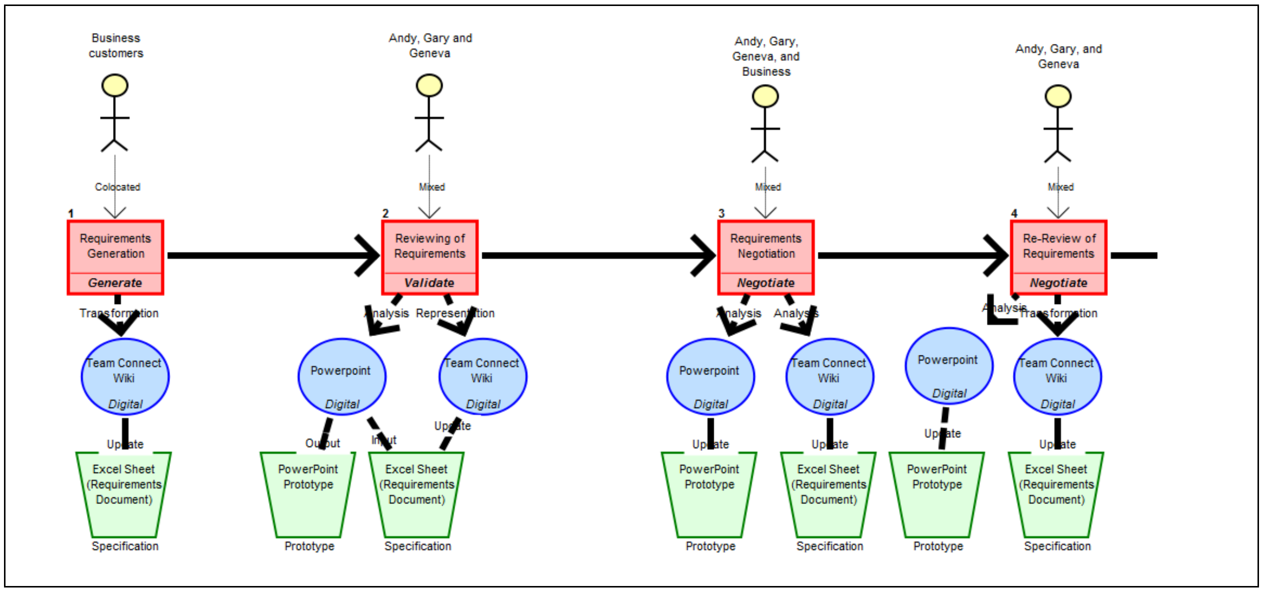


Figure 3. Development Process Model Visual Description

During these interviews, we included people in charge of methods, tools, and management strategies for these projects. Initially, we collected data on one waterfall project and one agile project and developed detailed workflow models of the routine composition and structure (see Figure 3). These were subsequently validated with the developers and managers. The detailed workflow sketches with information about the types of actors, activity types, design objects, and affordances used in carrying out each design activity were subsequently modified at this stage (see Figure 3) (Gaskin et al., 2014).

Typical projects contained between 200 and 1,000 activities with more than 40,000 total design elements. In 2011 and 2012, we added iteration objects to these models to improve coding procedures and simplify the visual layout of the models, which allowed us to quickly collect data when the process had a significant number of iterations. We validated each process model during the next field trip for all six software projects with their respective teams (see Appendix C for the visual workflows of all the projects).

3.3 Data Analysis and Coding

To address the general research questions, we broke them down into three detailed subquestions, which address the extent to which methods influenced how each activity within the process was carried out (so-called routine composition variation or configural variety) and the extent to which methods influenced how the activities were ordered, or so-called sequence variation or sequential variety (for a more detailed discussion about the types of variation in routines, see Gaskin et al., 2012). We address each specific subquestion below and describe how the data were analyzed.

3.3.1 At the Project Level, How Much Does a Method Induce Variation in Activities?

This subquestion seeks to generally assess the extent to which performative routines addressing a similar type of task (such as design) are similar to their ostensive specifications. We sought answers to this question through conducting three steps of analysis outlined below. We describe them briefly to show how we derived similarity/dissimilarity measures that helped us answer this research subquestion. Details for each step and the algorithms used are presented in Appendix D.

Step 1: Prepare and identify activity sequences. We divided the sequence data set into two data sets, BOM, representing waterfall projects, and LCM for agile projects. The first data set contained sequences of all three waterfall projects and the second data set contained sequences of all three agile projects. Overall, the two data sets were roughly comparable and contained 1,482 and 1,603 activities in waterfall and agile projects, respectively. The larger number of agile projects does not indicate larger projects but rather the presence of smaller steps and more iterations. Next, we identified specific activity types, such as generate, choose, and validate, and related design objects for each design activity to reveal the method-induced variation for each type of similar activity (see Appendix E for the list of all activity types). We assumed that the elements of design activities and design objects would capture most variation induced by design methods because people or settings are not controlled by the method (Royce, 1970; Cockburn & Highsmith, 2001).

Step 2: Cluster the sequences. To measure the dissimilarity between the activity sequences, we computed the Levenshtein distance (a metric for calculating the differences between two or more sequences using insertion and deletion costs) between the concatenated strings with the first three characters of the series of elements in a sequence (Lindberg et al., 2016). Assuming the cost for a single conversion is set to 1, the total cost of the Levenshtein distance between these sequences would be 2 (Abbott 1995). We calculated the distances between every sequence pair, i.e., the "pairwise distance," to form a distance matrix. Next, we clustered the design activities based on the similarity scores using k-medoid algorithms and used grounded theory to identify the designated themes of design activity in identified clusters (Kaufman & Rousseeuw, 1990; Studer, 2013). We chose k-medoids algorithms because this technique is more robust to noise and outliers than other clustering methods, such as k-means. This gives an average silhouette width (ASW) for each cluster based on the similarity. A value of ASW of close to 1 indicates a high degree of similarity between the sequences; a value of 0 indicates that the sequences are highly dissimilar (see Appendixes D and F for more details on the clustering).

Table 3. Ostensive Dimensions for Method Comparison

Design methods	Application	Management	Technical	Personnel
Agile	Agility, responsiveness	Tacit communications	Informal, simple designs	Collocated, thriving on chaos
Waterfall	Stability, predictability	Document-driven communications	Formal, complex designs	Distributed, thriving on order

Step 3: Calculate method-induced variation from silhouette width and ostensive correction. In carrying out this task, we were inspired by Boehm and Turner's (2003) model dimensions to compare agile and waterfall methods based on their ostensive aspects (Boehm & Turner, 2003). This resonates with our study goals and has a close parallel to the underlying organizational routine concept. Like the analysis of organizational routine literature, Boehm and Turner's analysis emphasizes two parts in devising methods: one leading to stability and another creating flexibility. They suggest that agile and waterfall methods are introduced to create either agility or stability, and their framework identifies four dimensions (see Table 3) on which the ostensive aspects of agile and waterfall method can be compared (Boehm & Turner, 2003). This provides a means to analyze the level at which studied performative routines align with those four dimensions of method. Next, we briefly describe each dimension as a baseline for comparing agile and waterfall activities and discuss the extent to which they align with respective ostensive dimensions.

It should be noted that according to Boehm and Turner's analysis, application refers to the application of the design method to either increase stability or instability, typically through higher degree of agility and responsiveness or control. Management refers to customer relations, project planning, control, or project communications that occur in projects. Technical refers to approaches to requirements, testing and development, and their articulation in design methods. Personnel refers to customer characteristics, developer characteristics, and the culture around which design activities are organized (Boehm & Turner 2003, p. 51-52).

Using these dimensions, we coded all activities in the agile and waterfall projects and assigned a rank to the clusters based on how well the performative activities matched with the ostensive specifications (see Appendix G for more details about coding and illustrative evidence.) For example, the "collective code monitoring" cluster has activities that increase agility by "keeping the developers on their toes"; hence, we coded this cluster with a rank of 1 on the application dimension and coded the other clusters based on the decreased rate of agility derived from qualitative data (see Appendix Table G1 for details on the ranks of the clusters). Similarly, we ranked other clusters on all four proposed dimensions according to waterfall or agile. Then, we calculated a composite rank, which in principle expresses how well the design activities purport the ostensive goals of the method. This allowed us to calculate an average for the overall method-induced variation of the design method. Because the silhouette width is still quite generic, the clustering analysis may not validly reflect why some sequences have been clustered. Hence, we introduced a correction factor for providing a more realistic method effect. A higher rank suggests that the observed activities in the sequences followed the ostensive principles more accurately for that dimension, and, to this end, we assigned a correction factor[3] of 1 for high-influence clusters and 0.1 for low-influence clusters. This corrected the average silhouette width obtained in the prior step with the correction factor per the formula below where *i* refers to the number of the cluster and *n* refers to the total number of clusters. The step introduced a corrected ostensive alignment score, which principally evaluates the effect of design method in shaping the observed design activities.

$$method - induced\ variation = \frac{1}{n} * \sum_{i=1}^{n} (average\ silhouette\ width)_i * (ostensive\ correction)_i$$

3.3.2 What Are the High-Level Differences Between Agile and Waterfall Projects in Terms of Their Ostensive Aspects?

To address this subquestion, we read the transcripts multiple times to derive and capture the meaning and nature of activities (refer to Appendix G for more details about coding and illustrative evidence). We used content analysis and related coding techniques to detect observed differences in activities by using the four dimensions of application, management, technical, and personnel. This provided additional evidence of actual method use and related differences and triangulated the findings with computational findings. This helped us identify how the two methods actually differ in the associated design practice, i.e., how much it is being guided by the ostensive dimension of the method (Stemler, 2001).

3.3.3 What is the Method-Induced Variation in Ordering Activities Including Their Level of Iteration in Agile and Waterfall Methods?

Order variance measures the extent to which the method influences the order of development activities and, specifically, the extent to which activity patterns repeat, or iterate, over time (Gaskin et al., 2012). For

[3] A correction factor is needed for understanding the real effect of design method on the project activities from random clustering of the sequences. If the correction factor is high, it indicates the clustering of sequences occurred due to the presence of ostensive principles.

calculating the level of repeated activities, we computed the proportion of unique activities (the activities that do not repeat in the overall sequence of activities) and then determined the unique activity ratio in relation to the overall project activity) (for more details about order variance/sequential variety, see Gaskin, 2012). For instance, for a set of activities in the following sequence A-B-C-A-B-D, the % of unique activities would be 33.3 (2/6 * 100), because there are only two unique activities, C and D (as highlighted). Based on this, we determined the repeated activities percentage to be 66.6% (=100-33.3). Using this measure, we computed the repeated activity percentage distribution across the agile and waterfall projects.

To detect order variance, we classified identified traces of activities (identified in Step 1 of the first question) and their sequences (i.e., computed observed permutations) into three categories. These categories helped us operationalize three facets of iteration in development activity: (1) no iteration, (2) presence of iteration (a sequence repeats itself in a straight sequence), and (3) iteration within iteration (a repeated sequence is included within a repeated sequence). We analyzed the relationships between these different states of iteration by creating Markov chains that would model transitions and transition probabilities between these states in each project and within similar types of projects. To build the Markov chains, we coded activities and their sequences into three categories to identify three levels of iteration within each project. We classified activities that did not iterate as "nonrecurring" states and classified activities that iterated into two types of states: simple recurring and embedded recurring iterations. Simple recurring iteration states involved activities that had a probability of repeating throughout the development process. We defined "embedded recurring" states as repeated activities nested within recurring activities as a special class of recurring state. Recent work on Markov chains shows that first-order Markov chains can have memory, use the concept of "partitions" instead of states, and suggest this as an appropriate way to model the Markov chains with memory. In other words, instead of purely calling a state open or closed, states can be recoded as partitions and can have more states (in our case, three partitions: nonrecurring, recurring, embedded recurring) capable of taking memory into account (for more on this, see Lacorata, Pasmanter, & Vulpiani, 2003).

To illustrate this coding, we present two scenarios of requirement-gathering activities. The first involves sequential steps of gathering requirements (like in waterfall projects), and the second involves a string of activities where gathering requirements occurs in parallel with the design activity (like in agile projects, see Table 4). In the first scenario, gathering requirements happens sequentially through the following activities: (A1) virgin data model creation, (A2) first round of gathering requirements, (A3) meeting to negotiate requirements, (A4) clarification of requirements in email, (A5) updating use cases. In this scenario, there is one iteration/repetition of a sequence of activities (3, 4, and 5) twice. We code these repeated activities as "recurring state (R)" (see Table 4, Scenario 1). The first two events of (1) virgin data model creation, and (2) first round of gathering requirements do not repeat themselves and are coded as a nonrecurring state (N).

In the second scenario, gathering requirements happens concomitantly with writing test cases, whereby two repeated activities (A6, writing test cases; A7, testing the use cases) are squeezed between activities A4 and A5 (see Table 4). In this case, activities A6 and A7 repeat within the larger repeated sequence (A3, A4, and A5), and this happens as part of the iterated sequence in the first scenario. Iteration is now embedded in a bigger iteration cycle, and this is called embedded recurring state (E). Herein all sequences of activities (A6, A7) are coded as embedded the recurring state E. (For more details, see Appendix D.) Next, we report our research findings.

Table 4. States of Iteration for Two Given Sequences of Activities

Scenario 1	
Nonrecurring state (N)	$\underline{\text{A1A2}}$ A3A4A5 A3A4A5
Recurring state (R)	A1A2 $\underline{\text{A3A4A5}}$ $\underline{\text{A3A4A5}}$
Scenario 2	
Nonrecurring state (N)	A1A2 A3A4A6A7A5 A3A4A6A7A5
Recurring state (R)	A1A2 $\underline{\text{A3A4}}$A6A7$\underline{\text{A5}}$ $\underline{\text{A3A4}}$A6A7$\underline{\text{A5}}$
Embedded recurring state (E)	A1A2 A3A4$\underline{\text{A6A7}}$A5 A3A4$\underline{\text{A6A7}}$A5

4 Findings

We address the subquestions outlined in the methods section and then report design routine variation in software processes across sampled methods to show overall variation in detected activity sequences. We analyze the method variation as induced by the method specifications qualitatively and, finally, discuss the effect of methods on observed activity-order variation.

4.1 At the Project Level, How Much Does a Method Induce Variation in Activities?

We leverage a novel computational technique proposed in the method section that calculates method-induced variation by taking into account ostensive and performative aspects of design activities. Using this procedure, we found that agile projects had an overall method-induced variation of 0.42 of the activities. This is slightly higher than that observed in the use of the waterfall method, which had about 0.40 of variations explained by the method. The reason for the high degree of variation in agile projects can be associated with higher conformance to ostensive aspects, especially in terms of increasing agility in the process.

We tabulated the amount of method-induced variation in the descending order in agile projects to highlight the influence of the ostensive aspects on different types of performative activities and their compositions (see Table 5; see Appendix G for more detail). As can be seen in Table 5, Clusters C1-C5 provide higher method-induced variation and reduce the dissimilarities in the types of activities performed at the cluster level. Clusters C1-C5 contain several families of iterative activities around coding, monitoring, and testing, and the variations in these clusters ranged from 0.8 to 0.37, indicating that a wide range of activities was performed within the limits of the ostensive guidelines. These activities were performed frequently to improve the project’s agility. This finding shows that agility and iteration are important facets that reduce the differences across design activities performed on a periodic basis. We also found that activities in clusters C6-C10 were less iterative and had a lesser degree of ostensive specification. The difference often emerged because of a high usage of the specific contextual IT artifact. The activities’ similarity with the ostensive element in these clusters ranged from 0.35 to 0.05.

We tabulated the amount of method-induced variation in the descending order in the waterfall method projects to identify and highlight the level of influence of the ostensive specification on the performed activities and their compositions (see Table 6). Clusters C1-C5 with high similarity contained several families of activities that center around planning, testing, and meetings. The variations in these clusters ranged from 0.97 to 0.40, indicating significant uniform compositional variance and similarity with the ostensive element. Most activities seek to increase stability and predictability of the design process. We found that the similarity of activities in Clusters C6-C10 ranged from 0.35 to 0.09. These clusters had fewer formal activities and contained activities such as prototyping, design sketches, and undocumented requirements. Such activities were not specified in the ostensive element of the method. One reason for the emergence of these types of behavior is associated with changes in design routine variations created by fitness and actor-related factors.

Table 5. Method-Induced Variation in Agile Methods

	Cluster name	**Method-induced variation**
C1	Code iteration	.8
C2	Collective code-monitoring	.71
C3	Test cycles	.63
C4	Pair debugging	.58
C5	Task delegation	.37
C6	Code promotion	.35
C7	Program testing	.27
C8	Code inspection	.25
C9	Test case generation	.2
C10	Use case scenarios	.05
	Average	.42

Table 6. Method-Induced Variation in Waterfall Methods

	Cluster name	Method-induced variation
C1	Planning through IT artifacts	.97
C2	Testing code	.61
C3	Meeting, testing, and releasing	.50
C4	Test, fix, and release	.44
C5	Status checking	.40
C6	Prereviewing code	.35
C7	Quality control	.26
C8	Architecting and validating	.19
C9	Use case-driven programming	.15
C10	Prototyping	.09
	Average	.40

Overall, we found that design routines in both design methods aligned relatively well with the ostensive approach. This was specifically pronounced across technical and application dimensions of the projects, which generated the requisite speed in the agile projects and sufficient control of the design target in the waterfall projects.

4.2 What are the High-Level Differences Between Agile and Waterfall Projects in Terms of Their Ostensive Aspects?

Here, we again address our second question, which seeks to understand whether there are differences in the respective impact of agile and waterfall methods across projects. This will be evaluated in terms of each method's impact on its application purpose and the management, technical, and personnel dimensions of the project.

4.2.1 Application

Application refers to the application purpose of the method to either increase artifact stability or instability allowing a higher degree of responsiveness. According to Boehm and Turner (2003), agile projects emphasize a higher degree of agility and are therefore different from waterfall projects, which seek stability and predictability. Our qualitative coding of the agile and waterfall interview data provides additional insights and reveals the extent to which these purposes were followed. Agile projects carried out multiple activities that increased agility by being responsive to continually changing requirements. For example, one developer emphasized that he could change the code on the fly and use tricks in the project environment to increase its agility. He mentioned that he "could change it right now and if someone's using that service, they're going to see my change. We have coding tricks to get around that, where you make a copy of it and you'd work on the copy."

In contrast, waterfall projects carried out more stable activities. For example, designers used technical inspections and prereview meetings to increase the stability of the artifact and reduce the defect rate at the end of each project stage. As the code went through multiple screenings, the defects were reduced, but this process consumed more time and reduced responsiveness. One project manager noted:

> *We had a premeeting. That was probably about a week in advance of that ... It was actually a technical inspection at that point where defects were recorded, identified, and recorded. I think the premeeting was to try and minimize the amount of defects that were generated from that, from our eyes and the design team's eyes.*

Overall, agile projects were less predictable and were prone to a higher proportion of trial-and-error and high-risk design activities. However, we conclude that both agile and waterfall methods aligned relatively well in terms of ostensive principles in the application purpose area.

4.2.2 Management

Management refers to how the method approaches customer relations, project planning, control, and related project communications (Boehm & Turner, 2003). According to Boehm and Turner (2003), agile projects emphasize a high degree of tacit knowledge and are different from waterfall projects, which demand extensive documentation for managing and dealing with stakeholder concerns. Our coding of project data provides insights into the extent to which these management principles were followed, i.e., on the detailed level of usage of documentation and related IT artifacts. Agile projects in the formative stages relied less on documents; the requirements were largely tacit and informal and were typically held in the minds of developers. In the later stages, the requirements were managed with increased attention to the explicit documentation, sometimes providing even more documentation than required. One developer noted:

> *We concentrate on writing more in detail requirements; it used to be just the one-pager from the customer, and we don't really have a lot of, like, ID-related stuff inside. Requirements in 1.8 are in detail, more concrete, so everybody can look at that, most of the people can look at that requirement and understand what's in it.*

Waterfall projects faced anomalies in managing documents. Even though the developers were expected to produce extensive documentation, there were sometimes deviations in this process. One offshore developer noted that sometimes the changes in the components were not tracked. When we asked about related documentation, he said:

> *And then in other ones you have like SharePoint to develop the components. Should there be additional where you write additional documentation? Or is there none of that. Not typically at that level. So you don't update the documents like which you would put into SharePoint? There's nowhere after this point where you update it? Yeah, unfortunately. The code is the master. What's worse than no documentation? Bad documentation.*

This showcases that both agile and waterfall methods typically had weaker alignment with the ostensive elements in the management dimension but still sought to align with expected ways of relating to project stakeholders.

4.2.3 Technical

Technical or technical design refers to specific approaches applied to requirements, testing, and development and how they are articulated in the design method (Boehm & Turner, 2003). According to Boehm and Turner (2003), agile projects are more informal and simpler compared to waterfall projects in this dimension. Our analysis showed that agile projects were less formal and engaged in many informal meetings that led to greater productivity and faster code development. The code constituted the primary artifact around which the design iterations took place. This reduced the generation of other (unnecessary) design artifacts. At the same time, the requirement generation followed an informal process. One project manager noted:

> *So, this is the Santa Claus process, right? Who's been naughty, who's been nice. Everyone puts in their wish list. Some are valid, some are priorities, some are must-haves. And that whole process of rooting through, that is an interactive, coming together and then dispositioning them and tossing them, deferring them or going.*

Waterfall projects tended to be more formal and carried out many meetings and interactions with users that were documented in formal protocols. For example, one developer suggested:

> *So there's a big portion from September to December of all . . . There's a whole dedicated user team testing, so not the QC team, but actual users, a whole of lot of them, dedicated in testing and trying to break it.*

This snippet showcases that waterfall projects were generally more formal and facilitated repeated interaction with users based on stated protocols.

4.2.4 Personnel

Personnel refers to customer characteristics, developer characteristics, and the culture around which design activities are organized in the method (Boehm & Turner, 2003). Cockburn (2007) emphasizes that in agile projects, it is important to have dedicated personnel with higher cognitive skills and that personnel with poor collaborative skills should be avoided. Our analysis shows that agile projects did not have a dedicated collocated customer (Cockburn, 2007). One developer mentioned, "the customers didn't understand what they wanted. That was the major hang-up on that." Agile projects often operate at the edge of chaos because of the lack of a fixed set of requirements. In contrast, waterfall projects operated with customers who were collocated, and part of the development took place in India. As a result, the process was less chaotic. In our case, agile projects were less aligned in the personnel dimension because they did not have a dedicated collocated customer. At the same time, the waterfall projects were more organized and aligned well with the ostensive personnel dimension of the method.

Overall, our analysis shows that both agile and waterfall projects aligned well with ostensive aspects in terms of the application and technical dimensions. The activities of both types of projects had less alignment in the management and personnel dimensions. This was largely due to uncertainty concerning how much to document and how to organize and coordinate between project members.

4.3 What Is the Method-Induced Variation in Ordering Activities Including Their Level of Iteration in Agile and Waterfall Methods?

To answer this question, we tabulated the percentage of distributions of the repeated activities in both agile and waterfall projects. As expected, the tally shows that agile projects contained a higher proportion of repeated activities (see Figure 4). We therefore conclude that, in general, agile projects were more iterative—that is, they repeated the same patterns of activities. Furthermore, LCM 1.8 project, the fourth version release of LCM, was the latest project and actually contained no singular, unique activities—in other words, the proportion of unique activities appeared to reduce over time. One reason for this is that the later projects could eliminate slack from implementation activities and reduced error rates.

Interestingly, we found that the waterfall projects were also iterative. The proportion of repeated activities in the waterfall projects ranged from 85% to 99.8% (Figure 5), indicating that the waterfall projects also had a tendency to significantly iterate over certain design elements. PADB 1.4 and BOMFI projects were the most iterative because of quality-related concerns and the extensive iterations across use cases and design options.

We also examined patterns of iterations across the projects to determine the levels of nonrecurring, recurring, or embedded recurring states in related processes. We found that four projects had only simple process structures—that is, they involved movements between nonrepeating moves (N) to repeating moves (R). Two of these projects followed waterfall methodology, whereas all agile projects had such simple iterative structures. Figure 5 shows that the PADB 1.4 and BOMFI waterfall projects had only simple structures. A common thread connecting these projects was that they all experienced similar planning environments, which resulted in the increased use of extensive planning with project management tools to reduce the need for complicated transitions and iterations. Furthermore, these two projects also displayed similarities in transitioning to an iterative state (R) (Figure 5)

Two agile projects had simple iterative structures (Figure 6). LCM 1.5-1.6 represented the first project and thus had higher complexity because of insufficient understanding of the initial requirements. Hence, the probability of going from nonrecurring to nonrecurring/recurring was equally split (50/50). However, in LCM 1.8, the probability of transitioning from nonrecurring to recurring was very low (0.06), indicating lower levels of process complexity. Overall, we notice that the probability of iterations in both the agile and waterfall projects remained roughly the same (0.98-0.99) across time, indicating that these methods enable a similar proportion of iterations. This was somewhat surprising in that agile processes are generally viewed as iterating more than waterfall processes. What was not surprising was that waterfall projects ran several iterations because of later challenges in tracing requirements and the need to implement related design decisions.

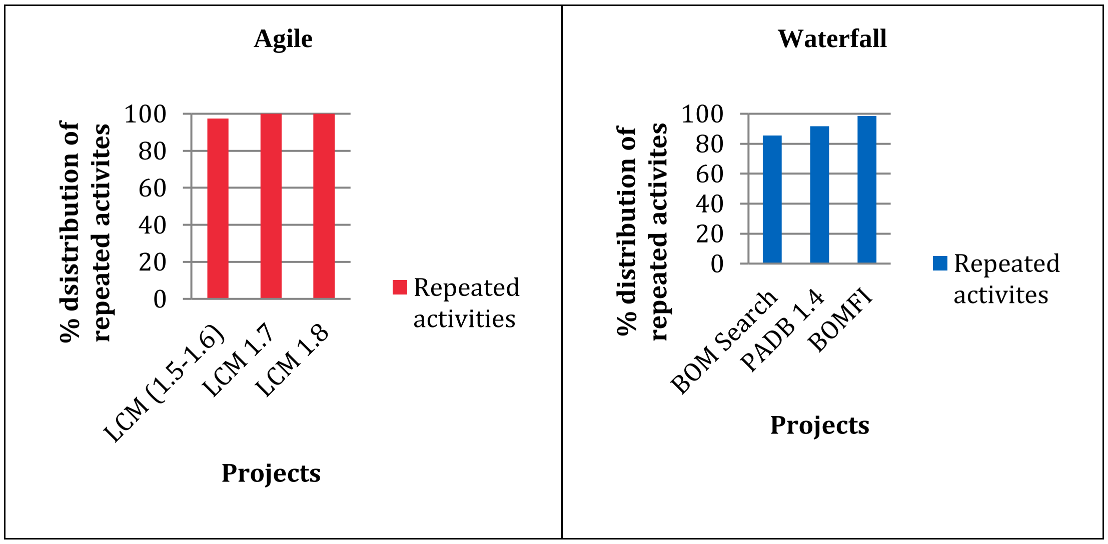


Figure 4. Distribution of Repeated Activties

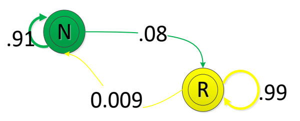
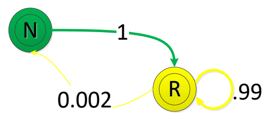
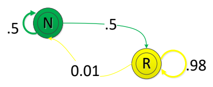
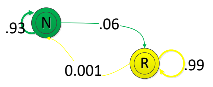
Waterfall projects	
	
PADB 1.4	BOMFI
Agile projects	
	
LCM (1.5-1.6)	LCM 1.8

Figure 5. Simple Iterative Structures

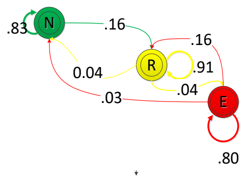
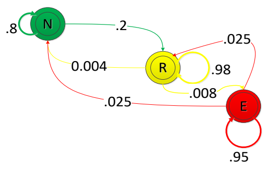
 .83 N .16 .16 0.04 R .91 .04 .03 E .80	 .8 N .2 .025 0.004 R .98 .008 .025 E .95
BOM Search (Waterfall)	LCM 1.7 (Agile)

Figure 6. Complex Iterative Structures

The projects that involved complex iterations (i.e., nested iterations) had a higher average probability of iterations. When nested iterations were present, this probability was 1.1 times higher in agile projects than in waterfall projects (Figure 6). Agile LCM 1.7 had more complex iterations due to the need for coordinating activities across several functional groups participating in the project (i.e., the business analyst group and developer group). Therefore, LCM 1.7 had an embedded iteration within the "requirements gathering" iteration for prototyping the collected requirements (Figure 6). The iteration was introduced strategically to minimize the discovery of bugs in the later phases, which could have resulted in a higher iteration probability in the project as a whole. Similarly, BOM Search waterfall project contained some complex iterations because of the significant overlap of development and testing activities in some stages. Overall, these findings suggest that agile and waterfall projects both iterate significantly but the reasons for this are different in the case of highly nested iterations.

5 Conclusions and Limitations

5.1 General Contributions

This study contributes to the large and well-established body of research on design methods and their effects. This work is unique in that it is carried out as a quasi-natural experiment, which allowed us to tease out the variation induced by the chosen method in design processes. To accomplish this, we adopted a novel theoretical lens of design processes as bundles of routines and used the construct of design routine variation as a conceptual means to detect how software development activities unfold and how variations in the activities can be explained by the chosen method.

Specifically, we examined how and to what extent performative manifestations of agile and waterfall methods differ and were able to attribute such observed differences to the presence of ostensive elements in these methods. We compared the profiles of design activities with the espoused profiles of chosen methods and compared the frequencies of repeated activities and structures of iterations with the probabilities to repeat the same set of activities between method conditions. Our findings suggest that the effect of using agile or waterfall methods is around 40% of performed design activities. The rest, about 60%, can be accounted for by other sources, such as method incompleteness, new or different fitness conditions, designers' skills and habits, and organizational noise. Both agile and waterfall methods had a stronger influence on the technical and application dimensions of the method than on the management and personnel dimensions. This suggests that the deviations from the methods multiply when designers face complex social or cognitive situations that are ambiguous, unclear, or difficult to coordinate. Third, unsurprisingly, the agile method involved more repetitive activities than the waterfall method, though the difference was smaller than expected. The probability of iterating in the agile method was 1.1 times higher than in the waterfall method when nested iterations were included. At the same time, the general iterations between the two methods remained the same when only simple iterations were included. Together, these findings advance our understanding of the effects of design methods and show that such effects are significant but probably less pronounced than often assumed in the literature. We also show that agile and waterfall processes actually mimic each other in many ways, with respect to iterations, and demonstrate how unique requirements are solved locally.

5.2 Contribution to Studies on Method Impact

A central contribution of the study is to empirically analyze differences in design routine variation between agile and waterfall projects. By doing so, we operationalize how to detect the effects of the ostensive dimension of the method on its related process enactment (Pentland, 2003). Prior research has already shown that, in specific settings, the performative dimension varies between agile and waterfall methods, although none of the studies have carried out strict comparisons (Mitchell & Seaman, 2009; Vidgen & Wang, 2009). These studies have focused primarily on qualitative, phenomenological differences, examining variations across design activities less systematically. One reason for this is that teasing out the influence of the ostensive dimension of a design method during its process enactment has been a challenging research task because of the lack of extensive process data and robust techniques to identify and capture such variance (Pentland, 2003). This study addresses some of these concerns. Our empirical results, though tentative and directional, support the general argument that the ostensive dimension of the method matters and indeed creates differences in the enactment of a design method. Furthermore, by developing measures of method-induced variance across two dimensions (composition and order) and using a detailed computational approach, we develop a way to tease out such variance. Our study provides a fine-grained analysis of how software design processes are orchestrated and shaped by design methods. Indeed, our analysis shows that there are true differences in how agile and waterfall methods influence development processes.

Our study also suggests that the current understanding of the effects of the ostensive dimension on the performative dimension is heading in the right direction: the effect of the method on performative routine variance is significant. At the beginning of the study, we expected the method effect to be more pronounced with the use of waterfall methods. However, we found that the different effects of the methods were not that far off from each other. This indicates that choosing any method matters to a certain extent in that it reduces and directs the compositional variance of the design activities. We also show that the order variation was different between waterfall and agile methods. We still cannot rule out the method effects in other facets of design, such as changes in the designer's attitudes and behaviors of other actors (such as users) because of the presence and reading of the ostensive specification. This is left for future study.

A second contribution of this work highlights the differences between agile and waterfall approaches in the application, management, technical, and personnel dimensions. This helps evaluate the extent to which the method used faithfully follows its espoused design principles. Our analysis suggests that both agile and waterfall methods follow the official design principles more thoroughly in application and technical dimensions, i.e., in terms of how the method-induced activities address the concern for agility or of formality and stability (Conboy, 2009). Our analysis shows, however, that designers often use coding tricks and create more dynamic environments to increase the agility of the processes. In our study, waterfall projects were more stable because they involved extensive quality and inspection tests that increased the stability of designs as well as the level of formality of the project activities.

We also found that agile and waterfall methods aligned less in the management and personnel aspects. For example, agile methods generally rely on informal knowledge exchanges that assume little or no documentation. In our study, this lack of documentation led to unexpected problems. Therefore, designers contextually modified the method over time by incorporating ostensive elements to increase the level of documentation. These findings suggest that in situ agile practices are often retrofitted and altered to support more explicit and formal method use. Even though the use of documentation might impair agility, such modified routines were often carried out to improve the final quality of the software. These findings illustrate that agile projects also showcase situational method adaptation, i.e., the ability to change and use project resources effectively and economically (Conboy, 2009).

A third contribution of our study provides novel empirical evidence for a well-established fact that the agile method involves more iterations than the waterfall method (Berente & Lyytinen, 2007). In practice, the waterfall method also demonstrated iterations. Designers carried out often repetitive activities akin to an agile process. However, we detected significant differences in the structures of iterations between agile and waterfall projects. This happened, in particular, when the iterations spanned multiple project tasks and involved several groups of project participants—that is, when iterations became embedded in multiple iterative cycles calling for more coordination. For example, when iterations were embedded, the average probability for iteration in agile and waterfall was 0.96 and 0.85 respectively. This suggests that agile projects tend to be a bit more iterative when there are multiple design groups, and they will repeat the same type of activities. At the same time, the average probability to iterate in agile and waterfall was similar even in situations when there were no iterations across groups and tasks. Our results suggest that agile and waterfall processes both involve iterations, although their frequency will vary according to method. Future research needs to elicit such differences by evaluating specific conditions that provoke or pacify alternative types of iterations.

5.3 Contribution to Studies on Software Processes

A considerable body of literature has addressed the differences between agile and waterfall methods in terms of cost, quality, and productivity (Lyytinen 1986; Lyytinen, 1987; Vidgen & Wang 2009). As expected, previous comparative studies found that agile developers spend less time in early stages managing requirements and produce more lines of code than their waterfall counterparts. Furthermore, agile developers are better at estimating effort and coming up with higher-quality products (Mitchell & Seaman, 2009). However, most of these studies were experimental and used students as study subjects, which raises some concerns about their external validity or real-world faithfulness. We need to question the extent to which can we directly translate such results to practice. Also, these studies did not use qualitative sampling techniques to explicate how the processes unfold contextually in real settings. In this regard, our study specifically contributes to the process side of software process research, which seeks to understand the real effects of method on process characteristics through fine-grained activity-level comparisons. We illustrate how much of the ostensive aspect of the methods are being followed in real-world settings. Our findings support the notion that methods are never fully followed and the designers need to be empowered to reflect on the evolving work practices in situ (Mathiassen and Stage 1990, Mathiassen and Purao 2002).

Future research may benefit from ethnographic studies that seek to understand the habits and skills of the agent and fitness-induced variations. Recent literature on open-source informalisms can provide a fruitful avenue to access the agency and fitness-based variations in agile and waterfall methods (Scacchi, 2002). Insight into the design rationale for the activities, artifacts, and affordances would likely shed light on the deeper issues around how design performance is orchestrated to reach an envisioned outcome (Conboy, Gleasure, & Cullina, 2015).

5.4 Contributions to Routine Research

Finally, we contribute to the routine literature by extending the ideas of ostensive and performative dimensions to reflect how design routine variations occur because of the presence of method, agency, fitness, and noise. Up to this point, only a few studies have discussed the emergence of performative routine variations, given the gulf between the ostensive and performative dimensions (Rerup & Feldman, 2011). The few exceptions are Turner and Rindova (2012), Jarzabkowski, Lê, and Feldman (2012) and Bucher and Langley (2016). These studies expose the microdynamics of the routine change based on agency and/or fitness-induced variation through concepts of “truce” and “reflective talk” (Zbaracki & Bergen, 2010; Jarzabkowski et al., 2012; Turner & Rindova, 2012; Bucher & Langley, 2016; Dittrich, Guérard, & Seidl, 2016). Our study complements these works by capturing variations that happen due to specific endogenous forces manifested in “iterations” (Patriotta & Gruber, 2015). We also highlight the effects of normative design methods, which have not been carefully addressed in prior studies. Accordingly, we observed design methods as the first source of performative variation in that they convey complex rule specifications that methods can use to exercise differential effects on performance. As yet, the research on routines has been silent about expanding the framework of ostensive and performative dimensions to study varying the effects of different sources of variation (Glaser, 2017). Our study complements the existing literature by demonstrating a way of analyzing and articulating variations within design routines and accounting for the strength of the relationship between ostensive and performative dimensions in the context of complex organizational work processes that follow rules or guidelines.

5.5 Implications for Practice

Our findings have practical value for software development organizations. Managers currently tend to be unaware of the extent to which and the dimensions in which ostensive principles shape the performance of process. For instance, project managers are expected to follow a design method that serves as a guiding template for providing timely software release (D'Adderio, 2014). Our analysis reveals that slippages from ostensive principles must be expected. We conclude that ostensive principles matter less than other aspects that influence projects. Thus, organizations should be cautious in investing in new methods and should adjust them only when needed.

5.6 Limitations and Future Research

Our study has several limitations. First, our research involved six projects in one firm. The sample remains limited, and the study should be considered exploratory. It is not necessarily generalizable across all organizations and software development situations. However, being the first study of its kind, it provides some new pathways that IS researchers could use to collaborate and seek additional insights about design processes. Future studies might ponder the effects on agency and evaluate how fitness in software development environments is achieved. The framework could be extended and generalized to reveal specific interconnections and variations in routines in different contexts (Dionysiou & Tsoukas, 2013). This study uses first-order Markov chains to compute order variance. In this regard, our analysis considers only the prior state to be important and relevant. Even though the sequences in the study have some memory, we do not apply higher-order Markov chains, as do some other works (Lacorata et al., 2003). In fact, some researchers suggest that this is not the appropriate way to model the Markov chain process with memory (for more details, see Lacorata et al., 2003). Because there is no clear understanding about how to model these processes for real-world settings, our Markov chain analysis has limitations regarding the estimation of the transition probabilities and how well they apply to other software development settings. However, because we are interested in comparing software design processes, the effects should be treated as illustrative of potential differences. Future research should focus on expanding the study to better understand the nuances of transitions and potential ways to extend Markov chain analyses and related design routine variations. In this regard, researchers should ask targeted questions regarding the extent to which actors and environments shape design situations given the level of risk involved (Schmidt, Lyytinen, & Mark Keil, 2001; Ramasubbu, Bharadwaj, & Tayi, 2015).

Acknowledgments

We are grateful for the openness and support of the people at Beta Corporation and are especially grateful to Dr. Alan Fisk for his continued support and enthusiasm. We are grateful to our colleagues Nick Berente and James Gaskin for their criticisms and comments. We also thank the senior editor and reviewers for constructive comments, which significantly improved the manuscript and added crispness to the argument. Finally, we acknowledge support of this research from the National Science Foundation through NSF Grant 0943157, Virtual Organizations Research Program and NSF Grant OCI-1121935, Virtual Organizations Research Program, and also Science Foundation Ireland grant 13/RC/2094.

References

Abbott, A. (1990). A primer on sequence methods. *Organization Science*, *1*(4), 375-392.

Abbott, A. (1995). Sequence analysis: New methods for old ideas. *Annual Review of Sociology*, *21*(1), 93-113.

Abrahamsson, P., Salo, O., Ronkainen J., & Warsta, J. (2002). *Agile software development methods: Review and analysis* (VTT publication 478, Espoo, Finland, 1-107).

Aldrich, H. E. & Pfeffer, J. (1976). Environments of organizations. *Annual Review of Sociology*,2 (1), 79-105.

Baskerville, R., Travis, J., & Truex, D. P. (1992). Systems without method: The impact of new technologies on information systems development projects. *Proceedings of the IFIP WG 8.2 Working Conference on the Impact of Computer Supported Technologies on Information Systems Development*.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., & Jeffries, R. (2001). Manifesto for agile software development. *The agile alliance*: 2002-2004, www.agilemanifesto.org.

Berente, N., & Lyytinen, K. (2007). What is being iterated? Reflections on iteration in information system engineering processes. In J. Krogstie, A. L. Opdahl, & S. Brinkkemper. *Conceptual Modelling in Information Systems Engineering* (pp. 261-278.). Springer.

Bijker, W. E. (1997). *Of bicycles, bakelites and bulbs: Toward a theory of sociotechnical change*. MIT Press.

Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, *35*(1), 64-69.

Boehm, B., & Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed*, Addison-Wesley.

Booch, G., Jacobson, I., & Rumbaugh, J. (1999). *The unified software development process.* Addison Wesley.

Bourdieu, P. (1990). *The logic of practice*. Stanford University Press.

Bucher, S., & Langley, A. (2016). The interplay of reflective and experimental spaces in interrupting and reorienting routine dynamics. *Organization Science*, *27*(3), 594-613.

Ciborra, C., Migliarese, P., & Romano, P. (1984). A methodological inquiry of organizational noise in sociotechnical systems. *Human Relations*, *37*(8), 565-588.

Cockburn, A. (2007). *Agile software development: the cooperative game*, Addison-Wesley Professional.

Cockburn, A., & Highsmith J. (2001). Agile software development, the people factor. *Computer*, 34(11), 131-133.

Cohen, M. D., & Bacdayan, P. (1994). Organizational routines are stored as procedural memory: Evidence from a laboratory study. *Organization Science*, *5*(4), 554-568.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research*, *20*(3), 329-354.

Conboy, K., Gleasure, R. & Cullina E. (2015). Agile design science research. *Proceedings of the International Conference on Design Science Research in Information Systems*.

D'Adderio, L. (2014). The replication dilemma unravelled: How organizations enact multiple goals in routine transfer. *Organization Science*, *25*(5), 1325-1350.

Davis, A. M., Bersoff, E. H., & Comer, E. R. (1988). A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*, *14*(10), 1453-1461.

Dionysiou, D. D., & Tsoukas, H. (2013). Understanding the (re) creation of routines from within: A symbolic interactionist perspective. *Academy of Management Review*, *38*(2), 181-205.

Dittrich, K., Guérard, S., & Seidl, D. (2016). Talking about routines: the role of reflective talk in routine change. *Organization Science*, *27*(3), 678-697.

Downey, H. K., & Slocum, J. W. (1975). Uncertainty: Measures, research, and sources of variation. *Academy of Management Journal*, *18*(3), 562-578.

Feldman, M. S., & Pentland, B. T. (2003). Reconceptualizing organizational routines as a source of flexibility and change. *Administrative Science Quarterly*, *48*(1), 94-121.

Felin, T., Foss, N. J., Heimeriks, K. H., & Madsen, T. L. (2012). Microfoundations of routines and capabilities: Individuals, processes, and structure. *Journal of Management Studies*, *49*(8), 1351-1374.

Feller, J., & Fitzgerald B. (2000). A framework analysis of the open source software development paradigm. *Proceedings of the International Conference on Information Systems*.

Fitzgerald, B. (1996). Formalized systems development methodologies: A critical perspective. *Information Systems Journal, 6*(1), 3-23.

Fitzgerald, B. (2000). Systems development methodologies: The problem of tenses. *Information Technology & People, 13*(3), 174-185.

Gaskin, J., Berente, N., Lyytinen, K., & Yoo, Y. (2014). Toward generalizable sociomaterial inquiry: A computational approach for zooming in and out of sociomaterial routines. *MIS Quarterly, 38*(3), 849-871.

Gaskin, J., Lyytinen, K. J., Yoo, Y., & Pentland, B. (2012). The effects of digital intensity on combinations of sequential and configural process variety. *Proceedings of the International Conference on Information Systems*.

Giddens, A. (1984). *The constitution of society: Introduction of the theory of structuration*. University of California Press.

Gilks, W. R., Richardson, S., & Spiegelhalter, D. J. (1996). *Markov chain Monte Carlo in practice*. Chapman & Hall/CRC.

Glaser, V. L. (2017). Design performances: How organizations inscribe artifacts to change routines. *Academy of Management Journal, 60*(6), 2126-2154.

Glass, R. L. (1991). *Software conflict: Essays on the art and science of software engineering*, Yourdon.

Gordon, V., & Bieman, J. (1993). Reported effects of rapid prototyping on industrial software quality. *Software Quality Journal, 2*(2), 93-108.

Hærem, T., Pentland, B. & Miller, K. (2015). Task complexity: Extending a core concept. *Academy of Management Review, 40*(3), 446-460.

Hirschheim, R. (2007). A comparison of five alternative approaches to information systems development. *Australasian Journal of Information Systems,* 5(1), 3-28.

Hirschheim, R. A., Klein, H. K., & Lyytinen, K. (1995). *Information systems development and data modeling: Conceptual and philosophical foundations*. Cambridge University Press.

Jarzabkowski, P. A., Lê, J. K., & Feldman, M. S. (2012). Toward a theory of coordinating: Creating coordinating mechanisms in practice. *Organization Science, 23*(4), 907-927.

Johnson, R. B., Onwuegbuzie, A. J., & Turner, L. A. (2007). Toward a definition of mixed methods research. *Journal of Mixed Methods Research, 1*(2), 112-133.

Kaufman, L., & Rousseeuw, P. J. (1990). *Finding groups in data: An introduction to cluster analysis.* Wiley.

Kumar, K., & Welke, R. J. (1992). *Methodology engineering R: A proposal for situation-specific methodology construction.* Wiley.

Lacorata, G., Pasmanter, R. A., & Vulpiani, A. (2003). Markov chain approach to a process with long-time memory. *Journal of Physical Oceanography, 33*(1), 293-298.

Laplante, P. A., & Neill, C. J. (2004). The demise of the waterfall model is imminent and other urban myths. *ACM Queue, 1*(10), 10-15.

Latour, B. (1986). The powers of association. In J. Law (Ed.), *Power action and belief: A new sociology of* knowledge (pp. 264-280). Routledge Kegan & Paul.

Leonardi, P. M. (2011). When flexible routines meet flexible technologies: Affordance, constraint, and the imbrication of human and material agencies. *MIS Quarterly, 35*(1), 147-167.

Levinthal, D., & Rerup, C. (2006). Crossing an apparent chasm: Bridging mindful and less-mindful perspectives on organizational learning. *Organization Science, 17*(4), 502-513.

Lindberg, A., Berente, N., Gaskin, J., & Lyytinen K. (2016). Coordinating interdependencies in online communities: A study of an open source software project. *Information Systems Research, 27*(4), 751-772.

Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., & Zelkowitz, M. (2002). Empirical findings in agile methods. In D. Wells and L. Williams (Eds.), *Extreme programming and agile methods: XP/Agile Universe 2002* (pp. 197-207). Springer.

Lyytinen, K. (1986). *Information systems development as social action: Framework and critical implications* (Doctoral dissertation, University of Jyvaskyla, Finland).

Lyytinen K., (1987). A taxonomic perspective of information systems development: Theoretical constructs and recommendations. In R. Boland and R. Hirschheim (Eds.), *Critical issues in information systems* (pp. 3-41). Wiley.

Markus, M. L., & Silver, M. S. (2008). A Foundation for the study of IT effects: A new look at DeSanctis and Poole's concepts of structural features and spirit. *Journal of the Association for Information Systems, 9*(10), 609-632.

Mathiassen, L., & Purao, S. (2002). Educating reflective systems developers. *Information Systems Journal, 12*(2), 81-102.

Mathiassen, L., & Stage, J. (1990). The principle of limited reduction in software design. *Information Technology & People, 6*(2-3), 171-185.

Mitchell, S. M., & Seaman, C. B. (2009). A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review. *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*.

Patriotta, G., & Gruber, D. A. (2015). Newsmaking and sensemaking: Navigating temporal transitions between planned and unexpected events. *Organization Science, 26*(6), 1574-1592.

Pentland, B. T. (2003). Sequential variety in work processes. *Organization Science*, *14*(5), 528-540.

Pentland, B. T., & Feldman, M. S. (2005). Organizational routines as a unit of analysis. *Industrial and Corporate Change, 14*(5), 793-815.

Petersen, K., Wohlin, C., & Baca, D. (2009). The waterfall model in large-scale development. *Product-focused software process improvement*. In F. Bomarius et al. (Eds.): *PROFES 2009, LNBIP 32* (pp. 386-400). Springer.

Ramasubbu, N., Bharadwaj, A., & Tayi, G. K. (2015). Software process diversity: Conceptualization, measurement, and analysis of impact on project performance. *MIS Quarterly, 39*(4), 787-807.

Rerup, C., & Feldman, M. S. (2011). Routines as a source of change in organizational schemata: The role of trial-and-error learning. *Academy of Management Journal, 54*(3), 577-610.

Roy, D. F. (1959). Banana time: Job satisfaction and informal interaction. *Human Organization, 18*(4), 158-168.

Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON.*

Russo, N. L., Wynekoop, J. L. & Walz, D. B (1995). The use and adaptation of system development methodologies. In M. Khosrowpour (Ed.), *Managing information and communications in a changing global environment* (p. 162). Idea Group Publishing.

Scacchi, W. (2002). Understanding the requirements for developing open source software systems. *IEE Proceedings-Software, 149*(1), 24-39.

Schmidt, R., Lyytinen, K., & Mark Keil, P. C. (2001). Identifying software project risks: An international Delphi study. *Journal of Management Information Systems, 17*(4), 5-36.

Schwaber, K. (1995). *Scrum development process*. Citeseer.

Shadish, W., Cook, T. & Campbell, D. (2002). *Experimental and quasi-experimental designs for generalized causal inference*. Houghton Mifflin.

Shoval, N., & Isaacson, M. (2007). Sequence alignment as a method for human activity analysis in space and time. *Annals of the Association of American Geographers, 97*(2), 282-297.

Smolander K., Tahvanainen V., and Lyytinen K.: How to combine methods and tools in practice- a field study. In B. Steinholtz, A. Sölvberg, & L. Bergman (Eds.), *Advanced information systems engineering* (pp. 195-214). Springer.

Sommerville, I. (1996). Software process models. *ACM Computing Surveys*, *28*(1), 269-271.

Stemler, S. (2001). An overview of content analysis. *Practical Assessment, Research & Evaluation, 7*(17), 137-146.

Studer, M. (2013). *WeightedCluster library manual: A practical guide to creating typologies of trajectories in the social sciences with R*, http://archive-ouverte.unige.ch/unige:78576.

Suchman, L. A. (1987). *Plans and situated actions: The problem of human-machine communication*, Cambridge University Press.

Turner, S. F., & Rindova, V. (2012). A balancing act: How organizations pursue consistency in routine functioning in the face of ongoing change. *Organization Science, 23*(1), 24-46.

Vidgen, R., & Wang, X. (2009). Coevolving systems and the organization of agile software development. *Information Systems Research, 20*(3), 355-376.

Yin, R. K. (2017). *Case study research and applications: Design and methods*. SAGE.

Zbaracki, M. J., & Bergen M. (2010). When truces collapse: A longitudinal study of price-adjustment routines. *Organization Science, 21*(5), 955-972.

Appendix A: Data Description

Table A1. Description of Waterfall Projects (BOM)

Name of the project	**Project description**
BOM Search	The Bill of Material (BOM) search project followed a traditional waterfall structure as dictated by Beta's life cycle development methodology that is founded on object-oriented data modeling, use cases, and derivation of a software design architecture using object-oriented design. The project was initiated in the first quarter of 2009 to enhance search in the BOM database and it lasted for about two years. It was relatively large in size (over 20 person-years) and involved 24 people working in two locations (United States and India). This BOM search project followed traditional phases of the waterfall methodology that involved gathering requirements, creating designs, coding and debugging, and testing the product sequentially with gate decisions in between (Davis, Bersoff, & Comer, 1988). The project also involved iterations within development and testing phases (Booch, Jacobson, & Rumbaugh, 1999). Overall, the BOM project followed the sequential phases as dictated by the waterfall methodology with partial overlaps.
Part Address Database (PADB) 1.4	PADB 1.4 is a continuity project to the PADB 1.2 project and was carried out in 2011 and 2012. With the growing requirements and scope, PADB 1.4 was kicked off to trace out the information that is embedded in a part number for detecting the type of parts used in the automobiles. This project used a phased approach inspired by waterfall principles for carrying out phases like inception, elaboration, transformation, transition with checkpoints in between stages. The interface that was built in this process was good functionally. However, schedule slippages occurred due to more development time and collaboration efforts. This project was distributed in the United States and India with 15 people in the project.
BOMFI	Bill of Material Foundation Integration (BOMFI) project was carried out in 2011 and 2012 to add additional databases and integrate them to the Bill of Material search database. Because the functionality was already in place, this project used waterfall principles to develop the product sequentially. Due to the smaller development effort required, this project contained 6 people and required around 12 person-years of overall effort. The project was carried out in the United States, Europe, and India.

Table A2. Description of Agile Projects (LCM)

Name of the project	**Project description**
LCM 1.5-1.6	This project addressed how the BOM database deals with engineering specification changes. The project has now been running for a few years and the software team creates a new release every three months with patches in between. We specifically investigated the design of the 1.5 and 1.6 releases referred to as "lightweight change management" or LCM. The 1.5 release began in September 2009 and went live with the release of 1.6 in January 2010. The development team chose to use an amalgamated Agile process for developing this application that was not strictly based on any particular method but was similar to the sprint phase of Scrum containing requirements, design, development, and testing. The room formation was adopted from the Team Room concept of Extreme Programming. The software progress and deadlines are reassessed daily, and changes are made as necessary. Thus, everyone involved is always knowledgeable about the status of the application and the deadlines.
LCM 1.7	LCM 1.7 was another version release of LCM carried out in 2010-2011. The project was carried out with five people distributed in the United States, Europe, and India. This project was implemented to change the back-end databases that integrate with the search for part-related information, namely, BOM. This was challenging, as the rapid changes in the agile back-end system caused struggles in the administrative areas in terms of coping with the change.
LCM 1.8	LCM 1.8 was the next version release of 1.7 and was carried out in a similar manner to that of LCM 1.7, though the scope of the project evolved through the backlogs and user stories that were originally created in the previsions releases. The project was carried out with five people distributed in the United States, Europe, and India.

Appendix B: Interview Protocol (Sample Questions)

1. Please begin by giving me a short history of your own career and how you came to be working with your present organization.
2. Please describe your current project, how it is organized, and what types of tools and artifacts you use or deploy? How much of this work is distributed in time and space? What are its main deliverables?
3. We are interested in various forms of information technologies that you use in your project. List all the main tools (both digital and nondigital) that you use for your design project. Tell me their main functions and how you use those functions to accomplish which goals.
    a. Can you tell us what one or two most important collaborative digital information technologies that your team has adopted recently?
    b. How did you come to adopt these tools?
    c. How are these tools currently being used in your projects and for which tasks?
4. We are interested in studying if and how the work practices and information technology use of your organization have changed based on your adoption of the tools you mentioned above.
    a. How has the nature of tasks in your project changed?
    b. How has the nature of collaboration in your project changed due to the adoption of new digital tools? Please give us specific examples of changes.
5. Have your design and development of these application platforms triggered the exploration of other digital or nondigital tools? Which ones?
6. Has your use of the digital tools affected the behaviors of other firms/stakeholders participating in your projects in any way?
7. How do you share, store, and coordinate various information related to your design project? How do you use digital tools in the process?
8. What were the main barriers, if any, in adopting these tools among different members on your project team at different sites involved?
    a. What were the main benefits for each group, individual, and tasks?
    b. Were there differences in the ways in which each group or individual had to work?
9. We are also interested in how these collaborative technologies relate to nondigital forms of collaboration.
    a. What has been the relationship between the use of digital and nondigital collaboration during this project? What is the proportion of each type of engagement?
    b. How has this relationship changed over the life of the projects?
10. How did your project members respond to the use of these digital tools?
    a. How did it compare to their “standard” or “traditional” way of working?
    b. How did they have to change the way they worked because of these tools?
11. How has the use of these tools affected your work and project management in the dimensions of cost, risks, quality, and work organization?
12. Next, we are going to analyze your current design processes and evaluate how digital tools are embedded in each step and phase of the task. Describe in chronological sequence a set of design tasks that you have carried out in this project since its start (can you check details from your calendar, email, activity log, etc.).

For each activity, please answer the following:

1. What were the tasks—what were their precedents, successors?
2. Was this part of a larger activity, and what was the purpose of the task?
3. What was your role in this task?
4. What were the deliverables and related design objects
5. Who was involved in this task and in what role (individual, meeting, etc.)? Where was the task located?
6. What tools were used?
7. How were those tools used?
8. How long did it take (duration)?

Appendix C: Descriptive Models of Agile and Waterfall Projects

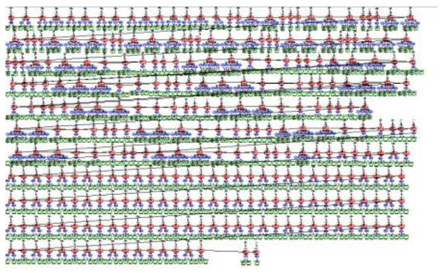
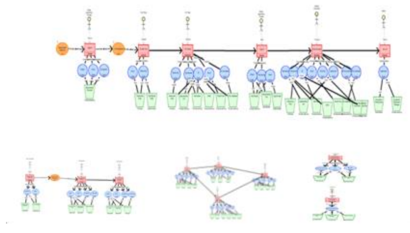
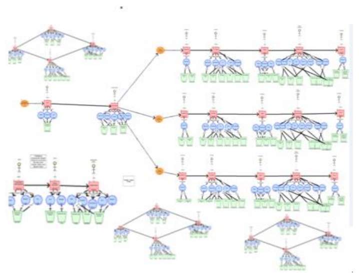
		
LCM 1.5-1.6	LCM 1.7	LCM 1.8
Notes: The first visual was developed without iterations and visually appears to contain more activities. LCM 1.5-1.6 had several massive iterations and resulted in the figure above. The later models were developed with the concept of iteration, which tremendously reduced the sketching of the process workflows.		

Figure C1. Process Models of LCM (Agile Projects)

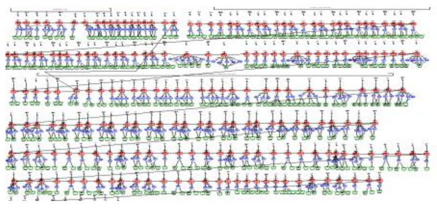
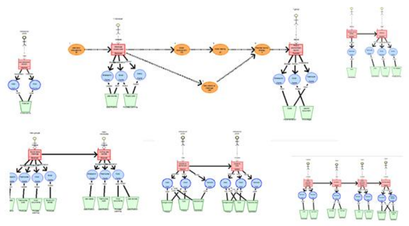
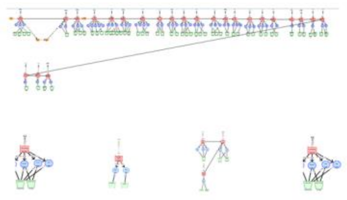
		
BOM Search	BOMFI	PADB 1.4
Notes: The first visual on BOM Search project also had iterations but had less iteration than LCM 1.5-1.6. The later waterfall models were developed with the concept of iteration as discussed above, which tremendously reduced the sketching of the process workflows.		

Figure C2. Process Models of BOM (Waterfall Projects)

Appendix D: Data Analysis Procedures

The data corpus had rich process descriptions and contained detailed data about the influence of methods, developers' expertise, and unexpected project conditions. Overall, the study was an embedded mixed-method study that used coding, content analysis, sequence analysis, and descriptive statistics as complementary techniques to detect and illustrate the design routine variation and its sources. The mixed-method design followed "complementarity" in using research methods. We sought elaboration, enhancement, illustration, and clarification of the results from one method with results from the other method (Johnson, Onwuegbuzie, & Turner, 2007, p 116) and expansion, and sought to expand the breadth and range of inquiry by using different methods for different inquiry questions (Johnson et al., 2007, p 116) to strengthen our analysis. After validating the process models, we used excel and R scripts to generate sequences that depict the actual design process. A typical sequence in the software process sequence contains seven elements: actor configuration, activity type, location, affordance type, tool type, data flow, and design object types (see Appendix E for full details about taxonomy) (Gaskin et al., 2014).

Method-induced variation is caused due to sequential nature of design activity or due to the usage and interplay of IT artifacts in design activity (Beck et al., 2001; Boehm, 2002; Cockburn, 2007). To analyze method-induced variation, we chose to analyze structural (sequential) and activity composition variation to capture the overall impact of methods on design performances. To this end, we carried out a structural analysis of the order of activities using Markov chains and state transition tables. This helped us understand the scope, frequency, and structure of iterations in agile and waterfall processes. One assumption behind the first-order Markov chain analysis is that it assumes that any given sequence of states or activities obey a Markov property—that is, the occurrence of any given event is only dependent on the immediately preceding event (Gilks, Richardson, & Spiegelhalter, 1996). Though not always reachable in software design, approximations based on first-order Markov models help detect structural differences in processes and their regularities. In our case, we used Markov chains to help identify distinct states of iteration and hence differentiate distinct states of iteration within agile and waterfall processes (Pentland, 2003). We then used sequence analysis to analyze the extent of variation in the composition of activities. At the same time, we relied on grounded theory and text mining techniques to identify and attribute the sources of such variation to methods.

To build the Markov chains, we coded the process activities and their sequences into three categories to operationalize three aspects of iteration. We classified activities that did not iterate as "nonrecurring" states. Those activities that had some probability of iterating (repeating) were classified into two types of "recurring" states: simple recurring and embedded recurring. Simple recurring states (hereafter "recurring states") involve activities that have some probability greater than 0 for repeating at some point throughout the development process. We further define "embedded recurring" states as subactivities nested within recurring activities as a special class of recurring state.

To clarify the coding, we elicited two scenarios where gathering requirements took place. The first involved sequential requirements gathering (i.e., waterfall), and the second involved a process in which requirements gathering paralleled the design activity (i.e., agile, see Table D1). In the first scenario, requirements gathering happened sequentially following the activities (A1) virgin data model creation, (A2) first round of gathering requirements, (A3) meeting to negotiate requirements, (A4) clarification of the requirements in e-mails, (A5) updating use cases. In this scenario, there was an iteration, i.e., a repetition of a set of events (3, 4, and 5) activities twice. We coded these repeating activities as "recurring state (R)" (see Table 2, Scenario 1). The first two events of (1) virgin data model creation, and (2) the first round of gathering requirements did not repeat themselves and were thus coded as a "nonrecurring state (N)."

Table D1. States of Iteration for Two Given Sequences of Activities

Scenario 1	
Nonrecurring state (N)	A1A2 A3A4A5 A3A4A5
Recurring state (R)	A1A2 A3A4A5 A3A4A5
Scenario 2	
Nonrecurring state (N)	A1A2 A3A4A6A7A5 A3A4A6A7A5
Recurring state (R)	A1A2 A3A4A6A7A5 A3A4A6A7A5
Embedded recurring state (E)	A1A2 A3A4A6A7A5 A3A4A6A7A5

Next, we describe the second scenario in which requirements gathering happened together with writing test cases. In this scenario, two new repeated activities (A6- writing test cases, A7- testing the use cases) were squeezed between Activities A4 and A5 (see Table 3). In this case, Activities A6 and A7 repeated within the larger repeated sequence (A3, A4, A5) in relation to the iterated sequence described in the first scenario. Hence, iteration here is embedded in a larger iteration cycle and it is thus called an "embedded recurring state (E)." In this scenario, Activities A6 and A7 were coded as an embedded recurring state, i.e., (E). (See Table 3, Scenario 2). We also show how transitions move from one state into another state in a specific set of activities (see Table D3).

Table D2. State Transitions

Input states	**Output states**		
	Nonrecurring	**Recurring**	**Embedded recurring**
Nonrecurring	1(Ex: A1-A2)	1(Ex: A2-A3)	0
Recurring	0	1 (Ex: A3-A4)	1(Ex: A3-A6)
Embedded recurring	0	1 (Ex: A7-A5)	1(Ex: A6-A7)
Notes: 0 indicates no possibility of transition input states to output states; 1 indicates the possibility of transition input states to output states			

Next, to analyze activity variations, we divided the sequence data set into two data sets: BOM for waterfall projects and LCM for agile projects. The first data set contained sequences of three waterfall projects, which included BOM Search, PADB 1.4, and BOMI projects. The second data set included LCM 1.5-1.6, 1.7, and 1.8 projects. Overall, the two data sets were roughly comparable, as they contained about 1,482 and 1,603 observations of activities in waterfall and agile projects, cumulatively.

To analyze the method-induced variation through ostensive rules, we selected a subset of the elements of the overall activity model and, for each activity, used information about its activity type and participating design objects. These two object elements from the activity model were selected because they tapped into the nature of design activity and identified involved design objects and their roles. We believe that this captures most of the variation induced by design methods (Royce, 1970; Cockburn & Highsmith 2001). Consider the following routine sequences A and B carried out in a waterfall project for (A) “gathering requirements,” and (B) for “status meetings.”

A: Generate/Specification/Specification/Prototype

B: Choose/Specification/Specification/Specification

For measuring the dissimilarity between these sequences, we computed the Levenshtein distance[4] between the concatenated strings containing the first three characters of the series of elements in a sequence. Assuming the cost for single conversion is set to 1, the total cost of Levenshtein distance between these sequences is 2 (Abbott, 1990). We calculated the distances between every sequence pair, called “pairwise distance,” to form a distance matrix. We then used k-medoids algorithm to partition the data sets into groups based on the value of pairwise distance scores between the sequences. We chose k-medoids algorithm because this technique is more robust to noise and outliers than other methods such as k-means (Kaufman and Rousseeuw 1990). For determining the number of clusters, we used the optimum average silhouette width (ASW), which seeks to increase the homogeneity of each cluster and ensures better validity of the identified clusters. Typically silhouette width ranges from -1 to 1 and ASW ranges from 0 to 1. Kaufman and Rousseeuw (1990) suggest that ASW > 0.71 for the identification of strong structures in the groups of data, and that silhouette width values are closer or nearer to 1 for well-classified observations. Based on these considerations, we extracted the clusters and silhouette width information for each observation. Further, we used grounded theory to systematically code activities in each cluster into respective key activity themes.

[4] Levenshtein distance is a metric that is used for calculating the differences between two or more sequences using insertion and deletion costs. Andrew Abbott (1990) popularized these concepts in social sciences with optimal matching algorithms that compute distance scores iteratively.

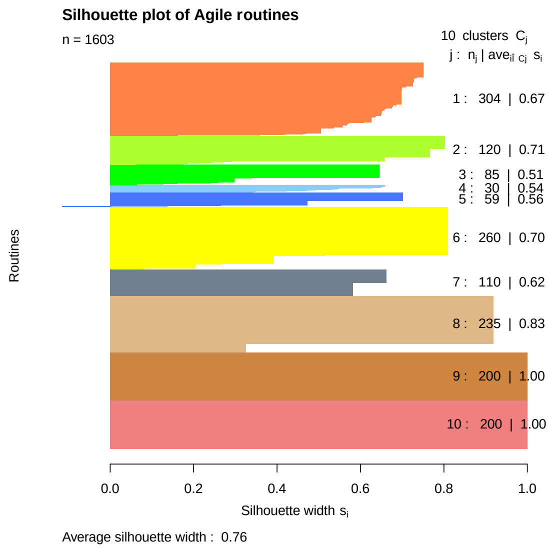
Appendix E: Taxonomy for Encoding Process Sequences

Table E1. State Transitions

Design component	**Items**	**Description**
Activity type refers to the purpose of the design activity.	Generate	Action-oriented planning and creativity-driven tasks such as brainstorming, coming up with plans, or producing something as a design
	Transfer	Transferring information or objects between people or locations
	Choose	Picking a correct or preferred option or answer. Coming to consensus
	Negotiate	Resolving policy and payoff conflicts
	Execute	Performing or executing a plan—producing an object according to a plan or a design
	Validate	Verifying quality and consistency
Actor configuration refers to the number and grouping of the actors involved in the activity.	One individual	Single individual
	One group	A group of individuals with a single functional purpose
	Many individuals	More than one individual, each with a separate functional purpose
	Many groups	More than one group, each with a separate functional purpose
	Individuals and groups	A mix of both individuals and groups, each with a separate functional purpose
Tool materiality refers to the material makeup of the tool being used for a particular design task.	Physical	The material nature of the functional aspects of the tool is physical, rather than digital. For example, the functional aspect of paper (ability to represent information) is physical
	Digital	The material nature of the functional aspects of the tool is digital, rather than physical. For example, a word processing document (ability to represent information) is digital
Tool affordance refers to “the possibilities for goal oriented action afforded by technical objects to a specified user group understood as relations between technical objects and users and understood as potentially necessary (but not necessary and sufficient) conditions for ‘appropriation moves’ (IT uses) and the consequences of IT use” (Markus & Silver, 2008, p. 622).	Representation	Functionality to enable the user to define, describe or change a definition or description of an object, relationship or process
	Analysis	Functionality that enables the user to explore, simulate, or evaluate alternate representations or models of objects, relationships or processes
	Transformation	Functionality that executes a significant planning or design task, thereby replacing or substituting for a human designer/planner
	Control	Functionality that enables the user to plan for and enforce rules, policies or priorities that will govern or restrict the activities of team members during the planning or design process
	Cooperative	Functionality that enables the user to exchange information with another individual(s) for the purpose of influencing (affecting) the concept, process or product of the planning/design team
	Support	Functionality and associated policy or procedures that determine the environment in which production and coordination technology will be applied to the planning and design process
	Infrastructure	Functionality standards that enable portability of skills, knowledge, procedures, or methods across planning or design processes
	Store	Functionality that allows information to be housed within a device
Activity location refers to where the design activity takes place.	Collocated	Actors are located in close proximity to each other at headquarters during the design activity
	Distributed	Actors are distributed during the design process
	Remote collocated	Actors, though located in close proximity to each other, are not at headquarters during the design activity
	Remote distributed	Actors are distributed and not at headquarters during the design activity
Design object type refers to the purpose of the design object being used as an input, being updated, or resulting as an output of a design activity.	Specification	The design object is instructions for design product parameters and constraints
	Design	The design object is a physical or digital prototype of part or the entirety of the intended eventual design product. This design object is used for further analysis and representation
	Implementation	The design object is actually used to complete, in part or whole, the intended eventual design product
	Process planning	The design object is instructions for future design activities
Tool-design object connection	Output	The data flow when the design object did not exist prior to the task, but was created during the task
	Input	The data flow existed prior to the task, but did not change during the task
	Update	The data flow existed prior to the task and did change
Notes: See Gaskin et al. (2014) Appendix A for more details about the taxonomy.		

Appendix F: Clustering in Method Solution and Examples

Table F1a. Silhouette Plot Clustering Solutions in Agile Projects

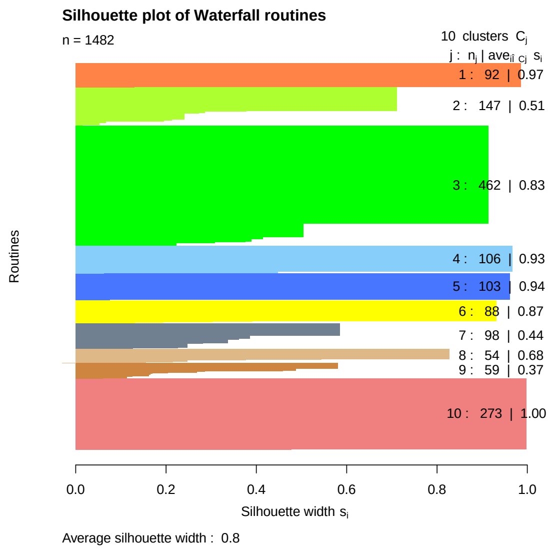


Notes:

Here n refers to the total number of observations, j refers to the serial number of the cluster, n_j refers to size of the cluster and s_i is the silhouette width of the cluster. Below are the names of the clusters for 1-10 in the picture on the left.

Cluster 1: Program testing
Cluster 2: Collective code monitoring
Cluster 3: Use case scenarios
Cluster 4: Code promotion
Cluster 5: Code inspection
Cluster 6: Test cycles
Cluster 7: Task delegation
Cluster 8: Pair debugging
Cluster 9: Code iteration
Cluster 10: Test case generation

Table F1b. Silhouette Plot Clustering Solutions in Waterfall Projects



Notes:

Here n refers to the total number of observations, j refers to the serial number of the cluster, n_j refers to size of the cluster and s_i is the silhouette width of the cluster. Below are the names of the clusters for 1-10 in the picture on the left.

Cluster 1: Planning through IT artifacts
Cluster 2: Use-case driven programming
Cluster 3: Meeting, testing, and releasing
Cluster 4: Prototyping
Cluster 5: Architecting and validating
Cluster 6: Test, fix and release
Cluster 7: Pre-reviewing code
Cluster 8: Testing code
Cluster 9: Quality control
Cluster 10: Status checking

Table F1. Examples of Activities in Each Cluster in LCM (Agile Methods)

Cluster no.	Name of the cluster	Size	Examples
1	Program testing	304	Use case validation, bug fixing
2	Collective code-monitoring	120	Coding, 10 AM (status) meeting
3	Use case scenarios	85	Test case writing, 4 PM (show and tell) meeting
4	Code promotion	30	9 AM status meeting, promoting code
5	Code inspection	59	Test case writing, QC testing
6	Test cycles	260	Status meeting, validate prototype
7	Task delegation	110	Morning meeting, afternoon meeting
8	Pair debugging	235	Small team meeting
9	Code iteration	200	Coding
10	Test case generation	200	Test case writing

Table F2. Examples of Activities in Each Cluster in BOM (Waterfall Methods)

Cluster no.	Name of the cluster	Size	Examples
1	Planning through IT artifacts	92	Roundtable meetings, developing project plan
2	Use case driven programming	147	Writing user stories, developing components
3	Meeting, testing, and releasing	462	Daily development standups, weekly status meeting
4	Prototyping	106	Generate raw data file/model
5	Architecting and validating	103	Validate the model, developing implementation model
6	Test, fix, and release	88	Clone environment development, Prod 2 launch
7	Prereviewing code	98	QC testing, premeeting, technical inspection
8	Testing code	54	User team testing, review by lead
9	Quality control	59	Clarification in emails communications, fix the defects
10	Status checking	273	Daily status meeting

Appendix G: Findings

To assess the performative variation of BOM and LCM projects, we computed clustering solutions to partition activity data into meaningful clusters. We used optimal average silhouette width (ASW) to partition data. This method maximizes the intercluster distances and minimizes the intracluster distance and can hence be considered a relatively robust solution than other competing clustering techniques, such as point bisreal correlation, Hubert's gamma, etc. (Studer 2013). To illustrate our clustering solutions, we plot average silhouette widths for different values of *k* (where *k* represents the number of clusters) (see Figures G3, G4). We obtained an optimum ASW of 0.8 and 0.76 in BOM and LCM projects at *k* = 10. Typically, ASW > 0.71 indicates an excellent split and indicates a high degree of homogeneity (Kaufman & Rousseeuw, 1990). It also indicates that a strong structure is found in the clustering solution.

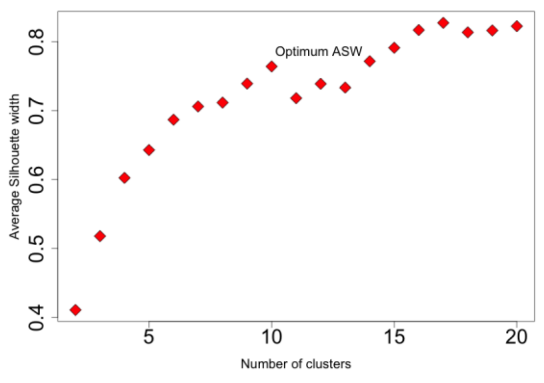


Figure G1. Agile Clustering Solutions

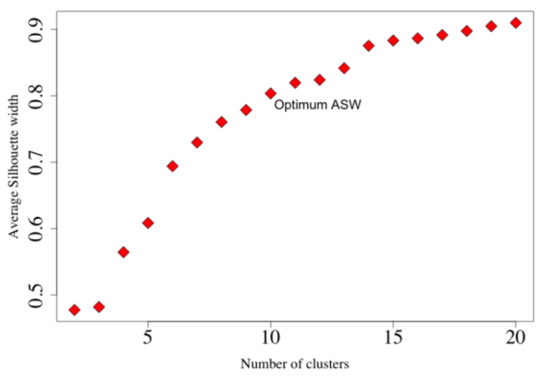


Figure G2. Waterfall Clustering Solutions

We ranked the clusters based on the ostensive aspects in terms of the application, technical, management, and personnel dimensions. See Tables G1 and G4 to find the ostensive correction of the cluster.

Table G1. Ranking of the Clusters Based on Dimensions of Ostensive Aspects

Cluster name	Application (agility and responsiveness)	Management (document driven, tacit)	Technical (informal vs. formal, simple vs. complex)	Personnel (collocation vs. distributed, thriving on chaos vs. order).	Overall rank
1. Program testing	8	7	5	5	7
2. Collective code-monitoring	1	3	2	6	1
3. Use case scenarios	9	8	9	10	10
4. Code promotion	7	4	10	1	6
5. Code inspection	4	10	8	9	8
6. Test cycles	2	5	3	3	2
7. Task delegation	5	1	4	7	5
8. Pair debugging	6	2	6	2	4
9. Code iteration	3	6	1	4	3
10. Test case generation	10	9	7	8	9

Table G2. Illustrative Quotations of the Agile Clustering Activities

Cluster name	**Examples**	**Illustrative quotations**
1. Program testing	Use case validation, bug fixing	*Instructors also used, for the big launch—for 1.0—where we had actual use cases and we had a dedicated—with them we have a dedicated QC person who tried to get as much knowledge as they can for that release. And then they'll create use cases for each.* *No, there was definitely like a lull period until users started using it and then there was sort of an onslaught of production bugs that came up.*
2. Collective code-monitoring	Coding, 10 AM (status) meeting	*That's one thing I should've mentioned earlier. 10:00 status, no matter what. Around the table. We also go through that task director. "Okay, these defects are still open," or "Are you working on these?" Because what they do it is they bid them to what release these defects are going to. So usually we know we are going to the next release, so we kind of keep going, "Okay, this is for the next release, this is for the next release, are you still making it?" Keeps us on our toes.* *Oh, the way Teamworks works is, you work on the code is this environment and it's live right as you're changing it. Yeah. It's interesting about Teamworks. Yeah. I could change it right now and if someone's using that service, they're going to see my change. We have coding tricks to get around that where you make a copy of it and you'd work on the copy. And when you're ready, you'd put it in the live version. We do stuff like that.*
3. Use-case scenarios	Test case writing, 4 PM (show and tell) meeting	*Yes. By the time we hit December, they were writing their test cases. Because we have to get all that code bundled up and put in their testing environment. Exactly. Exactly. That's exactly how it works. And then what period of time until it passed all the test cases? Um, let me see here. By the end, let's say before the break, that last week, there were only a few defects left.*
4. Code promotion	9 AM status meeting, promoting code	*So, Charlene goes in, and we did this every day for I think a week. We'd go in there, create a change, do this, and then we'd just kind of run an end-to-end test just to see if it went successful from the point at which the customer makes a change to the point at which you actually get an e-mail that it's been committed. So, we did that for a week. That was part of our daily status. "Oh, there's a bug. I see it. Here, let's fix it, try again. Oh, here's a bug. Let's work on that and we'll fix it tomorrow." So at certain points we'd stop, "Okay, let's see what we've got to do. Everyone go back and work on it. See if we can get it to work tomorrow." So, we'd go back and see if we could make those changes. And it probably took a good week to week and ½ to get a whole end-to-end test successful, where you could see the whole process stepping away. And the final complete was, "The email is sent. BOM changes are complete." Yay!*
5. Code inspection	Test case writing, QC testing	*Yeah. So, the formal QC happened I guess, maybe a little past the first week of December. We had dedicated the QC person to be launched … No, no. They're … cause they're just no room. They're in the next team room. So they're very close. They just walk over. They were also in those 11:00s as well. So they could get, have some knowledge what they're supposed to do. So they really know this stuff inside and out and by that point, has the QC person written all the test cases and everything? Yes. By the time we hit December, they were writing they're test cases. So their test cases are written and you think it's integrated and good? And you give it to them and they do all their test cases and they're letting you know?* *No, they were all still engaged and some of us were like, many of us were doing the testing but there were still some bug fixes and they were minor, but they didn't stop the process. Or, they were just changes to the coaches and stuff like—to the screens. So, we were kind of doing both. We were doing system integration testing as well as "Well, let's fix the screen to say this." So, we were doing the clean-up work as well as the system and the end testing.*
6. Test cycles	Status meeting, validate prototype	*So, from the PowerPoint, he starts mocking up some … not mocking up. He starts building some artifacts that are going to handle the changes correctly. And from that, he can show the users, "Here's the change*

		process. So, what we're going to do is cut it here and split it off here and do this and do that." *He also created some tables, which would house the changes. What we're doing is we take a change object and from that, we look into what pieces are important. In this case, it was cost. As an example, the customer said, "I don't want any cost changes going through for this vehicle line." So, he puts together a data model that can support that change. He puts together some screens that can support it. And then showed that back to Dave. And then from that.*
7. Task delegation	Morning meeting, afternoon meeting	*We had two days. We had a morning meeting and an afternoon meeting as well. So, there were two kinds or different types of meetings? So, your morning one would be issues and your afternoon one would be show and tell kind of? Yes, for the most part, but you do a mix of everything. Whoever had something to talk about, basically. I think our group was at least eight to ten people by the end of it. So, everyone had their own piece of code—their own section of it. So, whoever had problems—whoever had something to show.*
8. Pair debugging	Small team meeting	*And, it was under his guidance that the subteams met daily by phone. Charlene is in England. And proceeded to chew on the problem and break it down into smaller and smaller components that we were able to start building some artifacts to implement bits and pieces of it and push some sample data through and larger pieces of real data in sub-sections to see if the process was working.*
9. Code Iteration	Coding	*"Okay, by Friday, let's have our code ready. So, there's also a process of submitting your code, getting it approved. And all of the code is bundled in what we call modules. So, every piece of code in the module has to be ready in order for the code to be moved to the upper environment.*
10. Test case generation	Test case writing	*Yes. By the time we hit December, they were writing their test cases. Because we have to get all that code bundled up and put in their testing environment.*

Table G3: Ranking the Clusters Based on Dimensions of Ostensive Aspects

Cluster name	Application (agility and responsiveness)	Management (document driven, tacit)	Technical (informal vs. formal, simple vs. complex)	Personnel (collocation vs. distributed, thriving on chaos vs. order).	Overall rank
1. Planning through IT artifacts	3	1	3	1	1
2. Use case driven programming	5	10	2	9	8
3. Meeting, testing, and releasing	6	5	6	2	5
4. Prototyping	10	8	9	10	10
5. Architecting and validating	9	9	4	7	9
6. Test, fix, and release	3	7	5	8	6
7. Prereviewing code	1	2	10	5	3
8. Testing code	4	4	1	4	2
9. Quality control	2	3	8	6	4
10. Status checking	7	6	7	3	7

Table G4: Illustrative Quotations of Waterfall Clustering Activities

Cluster name	**Examples**	**Illustrative quotations**
1. Planning through IT artifacts	Roundtable meetings, developing project plan	*A lot of it was we had just a roundtable project and we walked through pretty much code.* *Microsoft Project, and we also used a tool called Clarity, and those project plans are loaded into Clarity, which produces a scorecard that management can recognize, and value and such.*
2. Use case-driven programming	Writing user stories, development of the components	*And when you had those user stories, then the idea was that you would write them for each use case a different implementation on a different platform in Java. Yeah. So that was through a SharePoint and then it wrote down all these user stories. There was a lot of project management with this.* *And then in other ones you have like SharePoint to develop the components. Should there be additional where you write additional documentation? Or is there none of that. Not typically at that level.*
3. Meeting, testing, and releasing	Daily development standups, weekly status meeting	*So the daily status meeting is done strictly with the development team and the testing, depending on the phase of the project, and the weekly status meeting is being done between me and my leads, and then another weekly meeting where we have a video call of the product backlog with the customer, and that's a weekly meeting. And then we can invite the customer or any other related customer to these weekly meetings when they choose later the business, we'll call them. So, the audience could vary, depending on the necessary…*
4. Prototyping	Generate raw data file/model	*Yeah, because the data relationships fundamentally were the ones we instituted with the U-BOM strategy, but what happened was, over a period of time they kept adding more and more attributes to AV-BOM that were not in the original U-BOM model. So there was a lot of catch-up on the attribute side to be done, understanding what had been done 'cause obviously at that point you're coming from a physical database and trying to work your way back through to you know logically what's there, conceptually what's there, and then how do we really want to represent that in the U-BOM world?*
5. Architecting and validating	Validate the model, developing implementation model	*Right. So what's driving behind the back of this, is as we have gone global and as we're bringing in the European development activities, the vehicles, there are some capabilities and functionalities that are needed in the European space that we haven't addressed, and so we're launching the software on vehicle programs and having some pretty significant problems with implementation.*
6. Test, fix, and release	Clone environment development, Prod 2 launch	*So after that first three months we, like I said, we got a clone environment of our existing development environment. All the code got imported there and we started developing from there.* *What does that mean, "practice launch"? So that strategy of launching, we get into the Prod II environment. We'd start from scratch with nothing and have a whole strategy of getting the DVAs involved loading data and then loading code. You'd have to do loads on the BOM-F side, so it's their launch strategy. So I think they took three days each.*
7. Prereviewing code	QC testing, premeeting, technical inspection	*There's also another quality inspection, I guess you'd call it, but that you'd say "do the design documents match the analytical documents," right? So there's like defects in that.*
8. Testing code	User team testing, review by lead	*So there's a big portion from September to December of all… There's a whole dedicated user team testing, so not the QC team, but actual users, a whole of lot of them, dedicated in testing and trying to break it.*

9. Quality control	Clarification in emails communications, fix the defects	*Yes. Firstly, I send an e-mail because they need to acknowledge it right away that this is the amount of discrepancy we found between the original use case and the design today. And then I'm actually updating my test cases based on this document. So this is my final version today. So, in case, tomorrow, if I'm logging in a defect or if I'm saying it didn't work the way you intend to, I reflect it in this document. Not to the one which we did before. So that was my clarification to them.*
10. Status checking	Daily status meeting	*So, I understand. So these latter types of meetings which you discuss more operational coordination and knowledge sharing within the project, whereas you have these bigger meetings with the clients like accepting the scope and other things, and then the other one was that now we are done with things, the inception phase, we can move to the next phase.*

About the Authors

Babu Veeresh Thummadi is a research fellow at Lero, the Science Foundation Ireland Research Centre for Software in University of Limerick located in Limerick, Ireland. He obtained his PhD from Case Western Reserve University. His main research interests lie at the intersection of software development, organizational routines, and open source.

Kalle Lyytinen (PhD, computer science, University of Jyväskylä; Dr. h.c. Umeå University, Copenhagen Business school, Lappeenranta University of Technology) is a Distinguished University Professor and Iris S. Wolstein Professor of Management Design at Case Western Reserve University, and a Distinguished Visiting Professor at Aalto University, Finland. He is among the top five IS scholars in terms of his h-index (87); he is the LEO Award recipient (2013), AIS Fellow (2004), and the former chair of IFIP WG 8.2 "Information Systems and Organizations" and Lorge Parnas Fellow. He has published ca. 400 refereed articles and edited or written over 30 books or special issues. He conducts research on digital innovation and transformation, design work, requirements in large systems, and the emergence of digital infrastructures.