12-15-2012

Iterations in software development processes: A comparison of agile and waterfall software development projects

Veeresh Thummadi

Kalle Lyytinen

Nicholas Berente

Iterations in Software Development Processes: A Comparison of Agile and Waterfall Software Development Projects[1]

B Veeresh Thummadi
Case western reserve university
vxt42@case.edu

Kalle Lyytinen
Case western reserve university
kjl13@case.edu

Nicholas Berente
University of Georgia
berente@uga.edu

ABSTRACT

Iteration is an essential element of software development processes. Software methodologies like agile and waterfall use the term 'iteration' in several different ways for improving either the quality or the functionality of the software. There are no studies, however, which have thoroughly analyzed and characterized these forms of iterations and their differences as they are enacted in organizations. In order to reveal these iterative forms we conducted a study of two mid-sized software development projects at a large global Fortune 100 corporation – one agile project and one waterfall project. Using advanced event sequence-analytic techniques based on detailed process data, our analysis reveals that agile and waterfall iterations differed in design and development phases due to different sources and types of iterations.

Keywords

Software process, iterations, agile, waterfall.

INTRODUCTION

There is a rich body of prescriptive literature touting the advantages of agile methodologies and offering guidance for managing iterations(Beck, Beedle et al. 2001; Cockburn and Highsmith 2001). However, even as the methods have been touted for more than a decade, and while some empirical research exists that compares elements of agile methodologies with those of other approaches(Boehm 2002; Larman 2004), there is little empirical work that compares the actual day-to-day activities across different espoused methodologies.

This is problematic, because we know that the 'espoused' method is never fully reflected in the performative practices of software design (i.e. method use). It is for this reason that ostensive (i.e. a method prescribed) method specifications have been criticized for their inability in effectively guiding the software development process (Lindvall, Basili et al. 2002; Damian and Chisan 2006; Dybå and Dingsøyr 2008). To wit, investigations into the impacts of using particular methods show significant variance in the way any methodology is being enacted across contexts (called "situational method adaptation"(Russo, Wynekoop et al. 1995; Sillitti, Ceschi et al. 2005). Therefore, though researchers have observed varying outcomes (such as quality and cost) associated with alternative methodologies, they have investigated less systematically *as-is performative practices* to understand what are the enacted differences between methodologies.

An important area of difference among methodologies involves the way they attend to the inevitable iterations that arise in the software development process. Certainly modern agile methods are iterative – they are typically touted for their attention to iteration(Larman 2004). However, so are more traditional methods such as waterfall(Royce 1970). Theoretical and empirical work both find that all software development is quite highly iterative, but this iteration is shaped differently across different methods(Thummadi, Shiv et al. 2011). Since all software development is iterative, it is not the presence of iteration that influences project outcomes, but instead the granularity, visibility,

[1] This research was supported by the National Science Foundation, VOSS Awards # 1121935, 0943157, 0943010

and locus of control in iterations that result in varying design outcomes(Berente and Lyytinen 2005; Berente and Lyytinen 2009). However, beyond pointing out some patterns of difference, there is no detailed, empirically-driven work aimed at detailing out what iterations mean *in practice* between agile methods and more traditional methods. To get at the differences between two broad software development methodologies (i.e., agile & waterfall) it is important to explore what different types of iterations there are, what are the sources for these different iterations, and what are the impacts of different forms of iteration. Thus we ask:

> *What are the sources, impacts, and types of iteration? Further, how do iterations differ across agile and waterfall projects?*

We address this question by first looking at the sources, impacts, and types of iteration evident in the literature. Then we investigate how one global organization adopted (and adapted) both the waterfall and agile methods for very similar software development projects. Specifically, we investigate the iterative nature of the enacted agile method and its use across the different phases of a software development project and how it compares to a traditional waterfall project. Since the two processes take place in the same organization and involve similar scopes, time frames, and team compositions, we treat them as a sort of natural experiment that is rife for theory generation(Shadish, Cook et al. 2002). We use a sequence analytic technique to investigate the differences, sources, types, and impacts of iteration across the two projects.

Overall, the study contributes to software design in two ways: 1) it compares Markov chain of iteration states between two processes as *to detect the probabilities of iteration and its scope and occurrence in a given process* and 2) it carries out an exploratory qualitative study *to detect sources, types, and impacts of iterations in agile and waterfall processes.* The remainder of the paper is organized as follows: in the next section, we review the literature on iteration, followed by a presentation of our method. Then we discuss the empirical study, describe the two projects, and report the findings. We conclude with a discussion of our findings.

ITERATION IN SOFTWARE DEVELOPMENT

In the early 1970's, software development methodologies were proposed to mitigate the failure of the large software development projects and the "spaghetti code" that resulted from unstructured development practices(Boehm 1988). Royce's (1970) waterfall method is perhaps the most well-known of these early structured methods (although there were a variety of others such as SDLC, see (Davis and Olson 1985). Although Royce explicitly characterized waterfall method in a way that allowed for iteration between stages, the method has since been typically conceived of as a sequential model indicating that once a stage was complete the development team has to move onto subsequent stages. In practice, of course, development activities are messy and non-linear in contrast to how the method conceives them (Parnas and Clements 1985); hence iteration is inevitable even when following a sequential concept of waterfall method(Dowson, Microelectronics et al. 1987). In response to what was found to be an unrealistic sequential view of methods, a variety of scholars have emphasized the inevitability of iteration in system development. They have proposed methods for explicitly dealing with this iteration(Wirth 1971; Basili and Turner 1975; Brooks 1975). This explicit accounting for iteration led to a variety of methodological innovations – including prototyping-based methods(Alavi 1984); rapid application development(Martin 1991); the spiral model (Boehm 1988); and a host of others. The most recent body of literature that pays particular attention to iterations in software processes are known as "agile" methods(Cockburn and Highsmith 2001).

Agile methodologies, at their very root, explicitly attend to the inevitability of iteration in software development and focus more on proper management of the iteration (Boehm 1988; Laplante and Neill 2004). Iterations in agile methods are managed using a variety of relatively novel techniques such as sprints, pair-programming, and time boxing – and they make also intensive use of traditional design techniques such as prototyping and refactoring (Miller 2001; Abrahamsson, Salo et al. 2002). However, the idea of iteration a multidimensional construct(Berente and Lyytinen 2005) and there is on-going debate about what "iteration" in software development is(Boehm 1988; Abrahamsson, Salo et al. 2002). In this section we would like to (re)define what iterations are and how they can be characterized specifically in the context of software development and related methodologies.

Iteration defined

The verb 'to iterate' is derived from a Latin word *itero* which means "to repeat" or "say again"(Riddle 1847) which emphasize repetitive behavior as an essential element in iteration(Berente and Lyytinen 2007). In the context of software development iteration can be considered as a (1) cyclical process of generating and testing the code (Beck 2000; Berente and Lyytinen 2007) or "repetitions of phases of development due to rework" or repetitive "subphases within a main phase"(Iivari and Koskela 1987; Berente and Lyytinen 2007) or any looping operations of activities occurring either in (2) prototyping or 3) problem-solving activities (Berente and Lyytinen 2007). Here we use the term "iteration" to refer to repetitive activities in software development so as not to confuse with the sprints of agile process(Berente and Lyytinen 2009)

Sources of iteration

The common sources of iterations in software process are 1) inadequacies in the specifications or assumptions; 2) changes in the requirements; 3) or errors in the design (Dowson 1987). The inadequacies in specifications about software design can be found in the initial specification documents like Software Requirements Specifications (SRS). The requirements in the initial stages of the development process are very uncertain as the "problem-solution"(Cheng and Atlee 2007) domain is quite blurred which results in iterations of the specifications artifacts. The next source of 'iterations' occur due to changes in the requirements. Requirements are never static and new information from the development process sheds light on the additional features that need to be catered in realizing the ultimate goals of the project(Smith and Eppinger 1997). The final source of 'iterations' occur due to errors in the code and design artifacts. Errors are by far the most common sources of iterations and hence software development teams equip quality control teams to improve the quality of the software.

Types of iteration

Iteration in software development can be classified into four categories based on the nature of the iteration (Humphrey 1986; Dowson 1987). *Functional iterations* are carried out for improving the functional elements of the design. Users realize the need for changes in the systems only when exposed to the new environment. Further inadequate understanding of the functional elements can lead to iterations early on in the process. Second types of iteration are *performance iterations* which are meant to improve the performance of the system. Developers and users realize the performance deficiencies only after the deployment stages. These iterations are unavoidable and can result in budget overruns and project delays. Thirdly *quality iterations* occur as a result of non-compliance to the requirements or the use cases. Code inspection and testing of the software can indicate the need for quality iterations. Fourth *combined/mixed iterations* occur as a result of combining performance/functional deficiencies and quality issues. Mixed iterations can result in complex management issues and large scale software failures(Dowson 1987; Humphrey 1994; Humphrey 1995).

Impact of iteration

Iterations play a vital role in software development process and their impact is inevitable in the form of consumption of resources and time (Dowson, Microelectronics et al. 1987; Dowson 1987; Eppinger 2001). Organizations encourage iterations in the form of feedbacks for identifying critical inconsistencies in the current software(Dowson 1987). After certain stage, iterations can reach a state when they can be called as "too much" as they don't significantly improve the existing software systems and are only marginally beneficial. *Good iterations* can boost the performances and productivity while *bad iterations* can totally derail the projects; hence bad iterations need to be weeded out for improving the success of software projects(Eppinger 2001). Further, iterations are perceived as "slack" when excessive rework and backtracking occurs(Dowson 1987). Hence, organizations encourage frequent iterations in the early stages of requirements in order to reduce bad iterations in the later stages of development.

Iteration in waterfall and agile methods

Agile and waterfall projects treat iterations quite differently. Especially agile method assumes iteration to be synonymous with the 'sprint' phase of SCRUM (see Fig.1). In agile methods, iterations are planned early on to cover a certain number of story points[2]. The uncompleted story points from previous iterations are carried on to the next iteration cycles.

[2] We use Cohn's (2011) definition for story point which refers to an "ideal day of work (i.e. a day without any interruptions whatsoever-no meetings, no emails, no phone calls etc) with fixed stories"

Waterfall on the other hand uses the term iteration to define refinement of the code after the completion of the phases by using feedback loops (see Fig 2)(Royce 1970; Brooks 1975). However, in practice both agile and waterfall exhibit iteration within the development life cycles in the form of sub phases wherein few set of sequential activities are repeated to meet certain ends(Iivari 1990; Berente and Lyytinen 2009). Unlike planned iterations, sub phase iterations are never planned for and occur due to uncertainties and complexities in the software development. Empirical studies have not tested the reasons, sources, characteristics for the iterations in the sub phases(Berente and Lyytinen 2009).

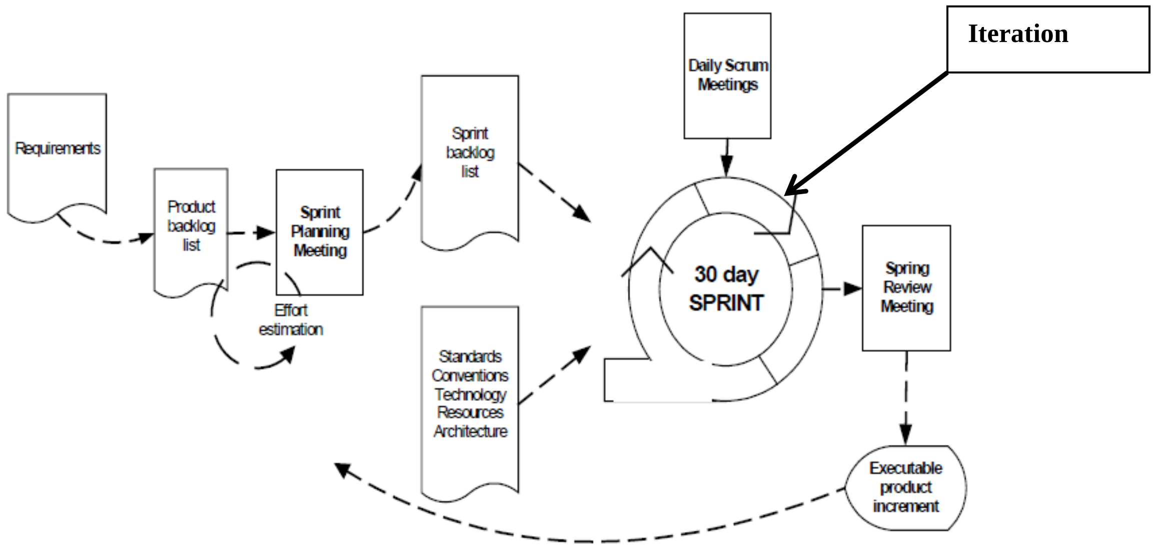

Figure 1: Iterations in agile process (Source: "Agile software development methods" P Abrahamsson 2002)

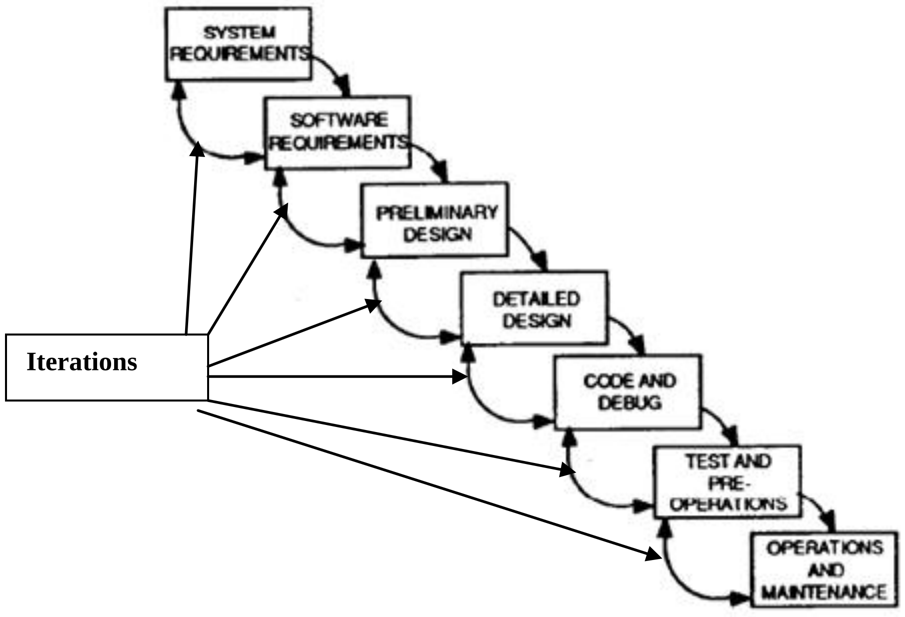

Figure 2. Iterations in waterfall process (Source: "Managing the development of large software systems" Royce, W. W. (1970))

Thus, while iterations are clearly different in agile vs. waterfall approaches, however, we do not understand how iterations manifest in practice. And what relevant types of iterations occur between these two approaches, and their drivers and their outcomes?

RESEARCH DESIGN

We studied two processes take place in the same organization and involve similar scopes, time frames, and team compositions. The only major difference is that one project followed an agile method and the other followed a waterfall-based method. Thus we view the study as a sort of "natural experiment" where many of the contextual issues are controlled but the explanatory variable differs(Shadish, Cook et al. 2002). Natural experiments are a sort of "quasi-experiment"(Shadish, Cook et al. 2002) and are well-suited to generation of theory.

Study sites and data collection

We collected process data from a large software development unit within a large multinational automobile manufacturing organization (referred to as "Beta") for specifically understanding how iterations occur in the case of software development. This unit focuses on developing and integrating the software that organizes product information and the associated processes for design and manufacturing. We selected one traditional waterfall project (BOM Search) and one agile project (LCM) of comparable scale and complexity that were developed during roughly the same time period using roughly the same sized team, and overlapping with many of the same artifacts and other infrastructural elements. For the waterfall project, we interviewed 7 of the 24 team members, and 5 of the 20 agile team members. We interviewed the managers of each team and also key personnel that those managers identified to give us a complete picture of the project. interviews were open-ended where we asked respondents to describe the initiation and sequence of events throughout the project. After the respondents introduced themselves and their role in the project, they typically were asked to describe the sequence of events with probes into "what happened next" and "describe that activity in more detail. What tools did you use? How did you use them? For what purpose?" Both projects involved a combination of distributed and co-located teams, but the agile project was primarily co-located, whereas the waterfall project was widely distributed – including internationally (between the U.S. and India). The waterfall team has a great deal of experience with the method, whereas the agile team is relatively new to agile methods. Although not a perfectly controlled experiment, the projects were selected with the idea that the main difference between the projects would be the explicit focus on using an agile methodology for one (LCM) and a waterfall method for the other (BOM), thus offering a possibility to conduct a sort of quasi-experiment or natural experiment (Shadish, Cook et al. 2002). We conducted in-depth interviews with project managers and team members and validated the process models using a thorough review process. Then we categorized these projects into phases for determining where similar types of activities happened (Kumar and Welke 1992; Abrahamsson, Warsta et al. 2003). We used two dimensions - 1) *the level of detail* reached in the activity based on design object to determine the start and end points for these phases and 2) *task coverage* to determine whether a specific task for the overall process was complete i.e. if it covers all the aspects of the phase-for determining the phases of the project (see Table 1&2) (Kumar and Welke 1992; Abrahamsson, Warsta et al. 2003).

BOM Search Project (Waterfall)

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 Bill-Of-Material (BOM) database and it lasted for about two years. It is relatively large in size (over 20 man years) and involved 24 people working in two locations (U.S. and India). Both these locations

The BOM project followed the phases of the prescribed in waterfall methodology enforced by the OEM that involved gathering requirements, creating designs, coding and testing the product sequentially with gate decisions in between. (See Table 1 &2 for details of the phases)

LCM Project (Agile)

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 "light change management" or LCM. The 1.5 release began in September of 2009 and went live with the release of 1.6 in January

2010. The project involved nearly 20 designers and a large number of lead users in OEM locations in the U.S. and Europe.

The development team chose to use an agile process for developing this application which was similar to the Sprint phase of Scrum containing: requirements, design, development and testing. The software progress and deadlines are assessed daily and changes are made as necessary.

Data analysis

We analyzed the interview data using a mixed-method approach i.e-use of qualitative (content analysis) and quantitative (markov chains) techniques simultaneously (Wheeldon).

We first used structural analysis using Markov chains and state transition tables as to understand the scope, frequency and structure of iteration in agile and waterfall processes (Pentland 2003). Markov methods have traditionally been used in engineering, economics and finance to predict the probability of an occurrence of a certain state. One assumption of the first order Markov chain is that it assumes that the given sequences obey a Markov property i.e., the occurrence of any given event is only dependent on the immediate preceding event(Gilks, Richardson et al. 1996). Though not always reachable in software design, approximations based on Markov models help detect structural differences in different processes and their regularities. In our case Markov chains help identify distinct states of iteration and hence are used to differentiate various states of iterations in agile and waterfall process(Feldman and Pentland 2003).

To build the Markov models we coded the process activities and their sequences into three categories to operationalize three different aspects of iteration. We classify activities that do not iterate as “non-recurring” states. Those activities that have some probability of iterating (repeating) are classified into two types of “recurring” states. Simple recurring states (here after, simply “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 sub-activities nested within recurring activities as a special class of recurring state. For explaining the coding states for the Markov chain, we use two set of scenarios where requirements gathering takes place. The first involves sequential requirements gathering (like waterfall) and the second involves requirements gathering parallel with the design activity (like agile, see Table 1). In the first scenario, requirements gathering happen sequentially following the activities (A1) virgin data model creation (A2) first round of gathering requirement (A3) meeting to negotiate requirement (A4) clarification of the requirements in emails (A5) updating use cases. In this scenario, there is an iteration i.e. repetition of a set of events <3, 4 and 5> two times. We can describe this iteration as a “recurring state (S_1).” The first two events of 1) virgin data model creation 2) first round of gathering requirement instead do not repeat themselves, and are thus coded as a “non-recurring state (S_0).”

Table 1. States of iteration for two given sequence of activities

Scenario 1	
Non-recurring state(S_0)	**A1A2** A3A4A5 A3A4A5
Recurring state (S_1)	A1A2 **A3A4A5** **A3A4A5**
Scenario 2	
Non-recurring state(S_0)	**A1A2** A3A4A6A7A5 A3A4A6A7A5
Recurring state (S_1)	A1A2 **A3A4**A6A7**A5** **A3A4**A6A7**A5**
Embedded recurring state(S_2)	A1A2 A3A4**A6A7**A5 A3A4**A6A7**A5

Next we can describe the second scenario where requirements gathering happen together with writing test cases where two repeated activities get squeezed between activities A4 and A5 (see Table 1). In this case, activities A6) writing test cases and A7) testing the use cases repeat within the larger repeated sequence and it happens in relation to the iterated sequence described in the first scenario. Hence iteration here is embedded in a bigger iteration cycle and hence it is called as "embedded recurring state (S_2)". In this scenario, all the activities A6 and A7 are coded as embedded recurring state i.e $S_{2.}$ (See Table 1). Further, we show how transitions occur from one state into another state in a specific set of activities (see Table 2).

Table 2: State transition table

Input states	**Output states**		
	Non-recurring	**Recurring**	**Embedded recurring**
Non-recurring	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*)*

0-Indicates no possibility of transition input states to output states
1-Indicates possibility of transition input states to output states

We then used the content analysis techniques to detect the sources, types, impact and effects of the iteration in two software processes. Content analysis is essential in determining the conditions that led to iterations (Gilks, Richardson et al. 1996; Duriau, Reger et al. 2007). To achieve this, we developed coding schema for analyzing iterations based on our research questions – what are the drivers, types, and impacts of iteration? Through literature review we elicited three sources for iteration (inadequacy, errors, and changes), four types of iterations (functional, quality, performance, and mixed), and two impacts of iteration (good and bad). Then we used content analysis to systematically code the interview data into few selective categories and themes. To this end we conducted content analysis of the interview transcripts using atlas.ti software by coding the categories from our schema established from the research questions (Stemler 2001).

FINDINGS

In this section we detail the findings of the two research questions. First we address the first research question –How do iterations differ across agile and waterfall projects and how are they structured?-using markov chains and transition tables. Then in the next section we address our next question- What are the sources, types, and impacts of iteration and in what ways do they differ across agile and waterfall-based methods? If they differ, what are the differences in sources and impacts and how they related to the methods used? – with the help of qualitative data

Differences in structure of iterations

Overall 7 subphases were iterated in waterfall (BOM) while agile (LCM) used only 4 sub phases for iterating(see Table 6). After looking at the requirements phase in agile and waterfall projects, we noticed that iterations were very similar in this phase. The frequency of iterations remained quite similar and surprisingly even the length of the iterated subphases remained very close. However, the main difference was that agile projects emphasized on performing newer activities in requirements than waterfall (see (S_0-S_0) values) in Table 6). We now show here an example of the calculation of probability for the requirements in BOM project. The sequences for the BOM project in requirements are as indicated below.

$$A0\ \underline{A1A1A1}\ \underline{A1A1A1}\ \underline{A1A1A1}\ \underline{A1A1A1}\ \underline{A1A1A1}$$

For a typical sequence of the transition for State 0 to State 0 is '0' as there is no transition from A0 to A0 in the given sequence. Similarly, the probability of transition from State 0 to State 1 is '1' (as there is only one possibility of transition i.e. from A0 to A1). In a similar way we computed the probability of other transitions (as shown in Table 3). The frequency tabulated in the table represents the number of times an iterated activity got repeated in the sequence. In the above mentioned case, the frequency is 5 times as the sequence A1A1A1 got repeated 5 times while the length of the iterated activity is 3. The length of time is in hypothetical "units."

Table 3. State transitions and granularity of subphases in agile (LCM) and waterfall (BOM) projects

Software design phases			State transitions							Granularity	
Phases of SDLC	**Design method used**	**Subphases in agile and waterfall**	**S_0-S_0**	**S_0-S_1**	**S_1-S_0**	**S_1-S_1**	**S_1-S_2**	**S_2-S_1**	**S_2-S_2**	**Frequency of iteration**	**Length of iteration**
Requirements	Agile	Requirement negotiation	0.25	0.25	0	0.5	0	0	0	6	2
	Waterfall	Use cases clarification	0	0.5	0	0.5	0	0	0	5	3
Design	Agile	Designing prototype	0.33	0.165	0	0.5	0	0	0	8	5
	Waterfall	Data model design	0.415	0.08	0	0.5	0	0	0	13	3
		Testing data model	0.33	0.165	0	0.5	0	0	0	3	6
		Informal inspection	0.425	0.07	0	0.5	0	0	0	3	5
Development	Agile	Code development and review	0	0.5	0	0.5	0	0	0	13	9
	Waterfall	Development of components	0.465	0.03	0	0.5	0	0	0	16	2
Testing	Agile	Bug fixing	0	0	0.5	0.5	0	0	0	60	2
	Waterfall	Development & testing	0	0	0	0.425	0.07	0.25	0.25	6,5*	6,2*
		Testing and release of components	0.415	0.08	0.03	0.465	0	0	0	2	8

Here * represents that there were two sub phases which got repeated. First activity consisted of 6 activities which repeated 6 times while second iterated sub phase consisted of two activities which repeated 5 times.

Next we looked at the design phases in agile and waterfall projects and noticed vast differences in iterations. Agile project had just one type of iteration for designing prototype while waterfall had three unique iterations for creating, testing and validating the data model. Further, the total numbers of activities iterated in waterfall were 97 and 40 in agile (see Figure 3). This is an important finding as it demonstrates agile methods conformance to ostensive specifications in creating "short bursts of design" and the exact opposite in waterfall i.e longer design phases. At the same time, we noticed that agile's designing prototype iteration and waterfall's testing data model iterations were similar in terms of activity composition and respective probabilities in transitioning (see Table 3).

Then we looked at the development phases in agile and waterfall projects and noticed that they are quite different in carrying development activities through iterations. Even though these projects emphasized on just one unique set of iterated activities for developing the code, the total number of iterated activities varied widely i.e agile 117 and 32 in waterfall (see Fig 3). Further, waterfall had a tendency to perform unique activities (see (S_0-S_0) values) in Table 3) while agile just carried iterated activities for developing the code.

Finally we compared the testing phases in agile and waterfall projects and noticed substantial differences in the way iterations are organized. Agile process contained one iteration which iterated more rapidly both of the waterfall iterations put together (see Table 3). Waterfall on the other hand contained complex iterations which involved subphases embedded together. The probability of finding the subphase in recurring state and embedded recurring state was .425 and .25 respectively which is indicative of the complexity and recurrence of iterations in the later stages of the development. Agile process emphasized on frequent and small iterations of the same type in testing phase while waterfall encouraged iterations to happen in different actor groups to reduce rework and relied on negotiation and clarification procedures (see Table 3).

Overall we noticed that agile process was more linear in terms of iterations. The frequency of iterations gradually increased over time in agile process while waterfall iterations "bulked up" in design phase. At the same time, we noticed that iterations in requirements phase were very similar in both these process.

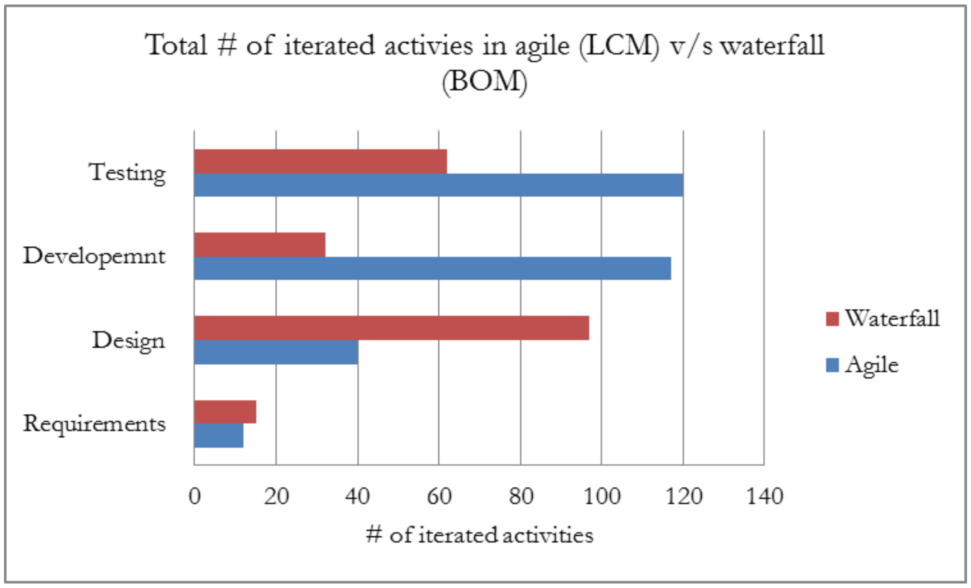

Figure 3. Iterated activities in agile (LCM) and waterfall (BOM) projects

CONCLUSION

From our analysis using state transitions tables in agile and waterfall process we identified the concentration of iteration efforts in both these process. The agile project relied on progressive iterations while waterfall relied on increasing the efforts on iterations in the design phase. This is indicative of the fact that waterfall emphasizes more discipline effort by confronting a big design phase to reduce iterations in the subsequent stages. Further iterations drastically differed in design, development in agile and waterfall process.

Agile process introduced quality control teams much later in the development and testing which led to a lot of bugs/defects in the later stages. In addressing these defects, agile team employed a "brute force strategy" and tried to address most of the defects by Quality Control (QC) teams. This strategy might have been useful for this agile project-(LCM-with 1.5 and 1.6 version releases)-as it minimizes defects in the future version releases like 1.7 and 1.8.

Practitioners can leverage the framework of iterations provided herin and begin thinking about iterations from a perspective of neither blanket iteration reduction. Iterations can be good in either a waterfall or agile context, but these are manifested in different activities between the methods – in waterfall iterations occur primarily on requirements and in waterfall in design and development.

REFERENCES

Abrahamsson, P., O. Salo, et al. (2002). "Agile software development methods." Relatório Técnico, Finlândia.

Abrahamsson, P., J. Warsta, et al. (2003). New directions on agile methods: a comparative analysis, IEEE.

Alavi, M. (1984). "An assessment of the prototyping approach to information systems development." Communications of the ACM **27**(6): 556-563.

Basili, V. R. and A. J. Turner (1975). "Iterative enhancement: A practical technique for software development." IEEE Transactions on Software Engineering **1**(4): 390-396.

Beck, K. (2000). Extreme programming explained: embrace change, Addison-Wesley Professional.

Beck, K., M. Beedle, et al. (2001). "Manifesto for agile software development." The Agile Alliance: 2002-2004.

Berente, N. and K. Lyytinen (2005). "Iteration in systems analysis and design: Cognitive processes and representational artifacts." Sprouts: Working Papers on Information Environments, Systems and Organizations **5**(4): 178-197.

Berente, N. and K. Lyytinen (2007). "What is being iterated? Reflections on iteration in information system engineering processes." Conceptual Modelling in Information Systems Engineering: 261-278.

Berente, N. and K. Lyytinen (2009). "ITERATION IN SYSTEMS ANALYSIS AND DESIGN." Systems analysis and design: techniques, methodologies, approaches, and architectures **15**: 37.

Berente, N. and K. Lyytinen (2009). ""Iteration in Systems Analysis and Design: Cognitive Processes and Representational Artifacts," in Chiang, Siau, & Hardgrave eds, Information Systems Analysis and Design: Techniques, Methodologies, Approaches, and Architectures, M.E. Sharpe, Inc. (AMIS Monograph Series, Volume 15), 2009."

Boehm, B. (1988). "A spiral model of software development and enhancement." Computer **21**(5): 61-72.

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

Boehm, B. W. (1988). "A spiral model of software development and enhancement." Computer **21**(5): 61-72.

Brooks, F. (1975). The mythical man-month: essays on software engineering, Pearson Education India.

Cheng, B. and J. Atlee (2007). Research directions in requirements engineering, IEEE Computer Society.

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

Damian, D. and J. Chisan (2006). "An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management." IEEE Transactions on Software Engineering: 433-453.

Davis, G. B. and M. H. Olson (1985). Management information systems: Conceptual foundations, structure and development, McGraw-Hill.

Dowson, J., Microelectronics, et al. (1987). Iteration on the software process, Microelectronics and Computer Technology Corporation (US).

Dowson, M. (1987). Iteration in the software process; review of the 3rd International Software Process Workshop, IEEE Computer Society Press.

Duriau, V. J., R. K. Reger, et al. (2007). "A content analysis of the content analysis literature in organization studies: Research themes, data sources, and methodological refinements." Organizational Research Methods **10**(1): 5-34.

Dybå, T. and T. Dingsøyr (2008). "Empirical studies of agile software development: A systematic review." Information and Software Technology **50**(9-10): 833-859.

Eppinger, S. D. (2001). "Innovation at the Speed of Information." harvard business review **79**(1): 149-158.

Farrell-Vinay, P. (1986). An approach to axioms of the iteration process. "Iterations in the Software Process" :proceedings of the 3rd International Software Process Workshop, IEEE.

Feldman, M. S. and B. T. Pentland (2003). "Reconceptualizing Organizational Routines as a Source of Flexibility and Change." Administrative Science Quarterly **48**(1): 94-121.

Gilks, W. R., S. Richardson, et al. (1996). Markov chain Monte Carlo in practice, Chapman & Hall/CRC.

Humphrey, W. S. (1986). Classes of process iteration Iteration in software process: proceedings of the 3rd international software process workshop, IEEE.

Humphrey, W. S. (1994). "Disciplined Software Engineering."

Humphrey, W. S. (1995). "Introducing the personal software process." Annals of Software Engineering **1**(1): 311-325.

Iivari, J. (1990). "Hierarchical spiral model for information system and software development. Part 1: theoretical background." Information and Software Technology **32**(6): 386-399.

Iivari, J. and E. Koskela (1987). "The PIOCO model for information systems design." MIS Quarterly: 401-419.

Kumar, K. and R. J. Welke (1992). Methodology Engineering R: a proposal for situation-specific methodology construction, John Wiley & Sons, Inc.

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

Larman, C. (2004). Agile and iterative development: a manager's guide, Addison-Wesley Professional.

Lindvall, M., V. Basili, et al. (2002). "Empirical findings in agile methods." Extreme Programming and Agile Methods—XP/Agile Universe 2002: 81-92.

Martin, J. (1991). Rapid application development, Macmillan Publishing Co., Inc.

Miller, G. G. (2001). The characteristics of agile software processes, Published by the IEEE Computer Society.

Parnas, D. and P. Clements (1985). "A rational design process: How and why to fake it." Formal Methods and Software Development: 80-100.

Pentland, B. T. (2003). "Sequential variety in work processes." Organization Science: 528-540.

Riddle, J. E. (1847). A progressive Latin-English vocabulary.

Royce, W. W. (1970). Managing the development of large software systems, Los Angeles.

Russo, N., J. Wynekoop, et al. (1995). "The use and adaptation of system development methodologies." Managing Information & Communications in a Changing Global Environment, Idea Group Publishing, PA.

Shadish, W., T. Cook, et al. (2002). "Experimental and quasi-experimental designs for generalized causal inference."

Shadish, W. R., T. D. Cook, et al. (2002). "Experimental and quasi-experimental designs for generalized causal inference."

Sillitti, A., M. Ceschi, et al. (2005). Managing uncertainty in requirements: a survey in documentation-driven and agile companies, IEEE.

Smith, R. P. and S. D. Eppinger (1997). "Identifying controlling features of engineering design iteration." Management Science **43**: 276-293.

Stemler, S. (2001). "An overview of content analysis." Practical assessment, research & evaluation **7**(17): 137-146.

Tashakkori, A. and C. Teddlie (2003). Handbook of mixed methods in social & behavioral research, Sage Publications, Inc.

Thummadi, B. V., O. Shiv, et al. (2011). Enacted software development routines based on watefall and agile methods: A socio-technical event sequence study. DESRIST.

Wheeldon, J. "Mapping Mixed Methods Research: Methods, Measures, and Meaning." Journal of Mixed Methods Research **4**(2): 87.

Wirth, N. (1971). "Program development by stepwise refinement." Communications of the ACM **14**(4): 221-227.