12-2016

Towards a software development methodology for projects in higher education institutions

Daniela Rivera Alvarado
Purdue University

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Daniela Rivera Alvarado

Entitled

TOWARDS A SOFTWARE DEVELOPMENT METHODOLOGY FOR PROJECTS IN HIGHER EDUCATION INSTITUTIONS

For the degree of Master of Science

Is approved by the final examining committee:

Alejandra J. Magana
Chair

John A. Springer

Mitchell L. Springer

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Alejandra J. Magana

Approved by: Jeffrey L Whitten 10/9/2016

Head of the Departmental Graduate Program Date

TOWARDS A SOFTWARE DEVELOPMENT METHODOLOGY FOR PROJECTS IN HIGHER EDUCATION INSTITUTIONS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Daniela Rivera Alvarado

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

December 2016

Purdue University

West Lafayette, Indiana

To my husband Juan, who has been my source of inspiration and encouragement throughout this journey. Thank you for always being there to support me, for believing in me, for your immense patience and continuous love.

To my parents Jorge and Grace, for sacrificing so many things to allow me become the person I am now and for your unconditional love.

To my sister Laura and my brothers Ernesto, Javier, Andres and Jorge, because your love and support has allowed me to keep on going every single day.

And last, but not least, to my daughter Cristina. Your only presence has made me a better person. For you is that I want to keep on growing and expanding, so I can be your guide though the years to come. Together we will explore the world and learn together. I love you to the moon and back.

ACKNOWLEDGMENTS

To my advisor, Alejandra Magana, for giving me the chance to be part of this program and for all to her understanding and support over the last five years. Thank you for never giving up on me.

To my friend, Jody Couch, for her constant words of encouragement and for always being there for me when I needed a person the most. I would have never completed this degree without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AT	Activity theory
CSR	Case study research
SDLC	Software development life cycle
SDM	Software development methodology

GLOSSARY

Activity theory	Cross-disciplinary theoretical framework that studies the actions of people, using an activity as the unit of analysis (Sam, 2012)
Case study research	Research strategy concentrated in the analysis and understanding of the dynamics found in single settings by reviewing single or multiple cases.(Eisenhardt, 1989)
Software development life cycle	Model created to follow a systematic and disciplinary approach in the creation of software solutions, meant to reduce the probability of chaos and failure and that takes a project solution from its inception to its retirement (Mahanti, Neogi, & Bhattacherjee, 2012)
Software development methodology	Recommended and proven way to successfully achieve the development of a system throughout the whole life cycle of a project (Vavpotic & Bajec, 2009)

Thematic analysis	Qualitative and inductive methodology that allows to identify, analyze and report patterns and overarching themes within data collected. (Braun & Clarke, 2006)

ABSTRACT

Rivera Alvarado, Daniela M.S., Purdue University, December 2016. Towards a Software Development Methodology for Projects in Higher Education Institutions. Major Professor: Alejandra J. Magana.

All educational institutions in the United States have certain particularities that differentiate them from many other public and private institutions. Some of these particularities include, among many others: academic year cycles that set very specific constraints and hard deadlines to the delivery of any tangible and intangible projects the institution is trying to accomplish; an always changing population of constituents that will be associated with the institution for a limited amount of time; and federal and state laws that are always evolving and that require the institutions to promptly act and adapt to fulfill the expectations set, in order to avoid severe lawsuits and fines.

As any other teams working in projects for educational institutions, software development teams are also heavily constrained by these particularities. This makes the adoption of Software Development Methodologies that perfectly fit other industries a daunting challenge, if not almost impossible, for these teams. Software development teams in higher education are always in the need of finding a way to adapt to these challenges and efficiently perform their projects in order to address the rapid changes occurring not only in the education sector, but also in the technology industry in general.

The purpose of the research in this thesis was to identify opportunities and challenges of software development methodologies used in higher education and to recommend a software development methodology to be used by software development teams working for those institutions.

CHAPTER 1. INTRODUCTION

The purpose of this research was to identify opportunities and challenges of software development methodologies used in higher education and to recommend a software development methodology to be used by software development teams working for those institutions. In this chapter the author presents the scope and significance of this project, followed by its research question and the different assumptions, limitations and delimitations that were put under consideration.

1.1 Scope

All educational institutions in the United States have certain particularities that differentiate them from many other public and private institutions. Some of these particularities include, among many others, (a) academic year cycles that set very specific constraints and hard deadlines to the delivery of any tangible and intangible projects the institution is trying to accomplish; (b) an always changing population of constituents that will be associated with the institution for a limited amount of time; and (c) federal and state laws that are always evolving and that require the institutions to promptly act and adapt to fulfill the expectations set, in order to avoid severe lawsuits and fines.

As any other teams working in projects for educational institutions, software development teams are also heavily constrained by these particularities. This makes the adoption of Software Development Methodologies (SDMs) that perfectly fit other industries a daunting challenge, if not almost impossible, for these teams. Software development teams in higher education are always in the need of finding a way to adapt to these challenges and efficiently perform their projects in order to

address the rapid changes occurring not only in the education sector, but also in the technology industry in general.

Software, as indicated by Holcombe (2008), is an essential element in the success of many businesses and organizations, which poses extra pressure in software development teams to deliver good quality applications in a minimum amount of time. In this research, the author tried to understand how the particularities of educational institutions affect software development teams at Purdue University, a land-grant higher education institution located in West Lafayette, Indiana. To accomplish this and while using a qualitative approach based on a case study research (CSR), a total of ten different interviews to software development team managers and developers were performed. These interviews served to gather information about the advantages and challenges these people face in their software development life cycles (SLDCs) while developing applications for the institution, as well as information about the methodologies their teams use to accomplish their projects.

After the data was collected, the information obtained was analyzed using a thematic analysis qualitative methodology. At the same time and based on existing literature, a review of known software development methodologies was done, where advantages and disadvantages were determined for each, resulting in an instrument that was later used by the author to analyze the themes identified from the interview data.

With the results obtained from this analysis and with the help of Activity Theory (AT), the author developed a proposed software development methodology that will help to address the challenges most commonly identified during data gathering, while incorporating the advantages of existing software development methodologies, helping to facilitate its easy adoption by software development teams in higher education.

1.2 Significance

Higher education institutions usually have one or more software development teams serving their in-house software development needs. Unfortunately, not much research has been done to explore the challenges these software development teams face on a day-to-day basis, given the particularities that surround higher education, and the restrictions these environments impose. In addition to this, not much research has been done either to determine which and how software development methodologies could better adapt to these particularities and constraints in order to make software development teams more effective and efficient in the delivery of their projects, within scope, time and budget.

The main goal of this research was to analyze the challenges software development teams currently face within their institutional environments and determine the best way to address them. Then, with the help of the AT theoretical framework, propose a methodology that could be adopted by any software development team in higher education institutions that would help them to easily overcome those challenges.

1.3 Research Questions

The following research questions are answered by this research project:

- How are software development methodologies currently being used by software development teams working for higher education institutions?
- How are these methodologies supporting or limiting the team's performance and outcomes?

1.4 Assumptions

The following assumptions were identified as key components and big influencers of this research:

- Based on the literature reviewed, a software development methodology that addresses the specific challenges and needs of software development teams creating applications for higher education institutions is yet to be developed.
- The participant subjects of this study provided an unbiased opinion on the opportunities and challenges their teams experience while developing software for higher education, and their answers were not affected by whether or not they knew or had interacted with the author of this research before.
- The data obtained through the collection methods was meaningful and it allowed the author to draw enough information to answer the questions posed as part of this research.
- The instrument used in the assessment of existing software development methodologies was properly designed and all results obtained were valid and meaningful.
- An IRB approval has been received from the Office of the Vice President for Research at Purdue University, that allowed the use of human subjects (in this case the managers and developers of software development teams at Purdue University) to participate in this research study.
- A total of five managers and five developers of software development teams at Purdue University demonstrated interest in participating in this research study.
- All participants had experience or had been exposed to at least one software development methodology and were familiar with the concept.

1.5 Limitations

Similar to the assumptions, the following limitations were identified as key players for the successful completion of this research:

- A qualitative study, following a case study research approach was selected as the best methodology for this research. This allowed the author to understand in depth, by collecting verbal descriptions, the challenges faced by software development teams creating systems for higher education institutions and it helped to develop a methodology that would address these challenges and needs.
- AT was selected as the theoretical framework to be utilized to sustain the creation of the software development methodology to be developed in this research project.
- CSR was selected as the methodological framework to be utilized to obtain the data to be utilized in the creation of the software development methodology to be developed in this research project.
- The number of participants for this study was limited to a total of five software development managers and five software developers that were part of teams developing applications for Purdue University.
- The software development methodology to be created in this research project would be based on a combination of other existing methodologies.

1.6 Delimitations

Similar to the assumptions and limitations, this research was performed by acknowledging the following delimitations:

- No software development teams from institutions other than Purdue were included in the data collection phase of this research.
- This research project does not set the expectation for Purdue University to adopt the software development methodology proposed as the only methodology to be used in the development of software applications.

- An implementation to test the effectiveness of the software development methodology developed in this study was not included as part of this research project.
- This research study cannot be considered a final solution to the different collaboration and organizational problems that can be found during the software development cycles of teams in higher education.

1.7 Summary

This chapter provided a review of the scope, significance, research question, assumptions, limitations, delimitations, definitions, and other background information for this research project. In the next chapter, a review of the literature relevant to the project is presented.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

In this chapter the author provides a review of the main different areas of argumentation that support this research project. The review begins by outlining how software development relates to higher education in general, putting some attention into the challenges associated with it. The review then proceeds to highlight the research that has been done to address software development for this higher education in general.

Following this review, the chapter continues with an examination of SDMs, which are frameworks that are used by software development teams to plan and execute their projects. The goal is to deliver the systems in time, within budget, full in scope, and with the expected quality. Of importance is analyzing why software development methodologies are needed and the current gaps in research related to the current methodologies that could easily adapt to a specific area like higher education.

In a subsection, this chapter talks about the most well-known software development methodologies used by the industry nowadays, in order to provide some context of the different approaches that have been followed by software development teams.

The chapter finishes with the comparison of existing SDMs, utilizing two different but complementing approaches found in previous studies done in this research area.

2.1 Software Development in Higher Education

The nature of higher education is always changing (Kennedy, 1998). New students join higher education institutions every year, coming from very diverse

backgrounds, interests and motivations. For this reason, software has moved from being a fringe asset in higher education to being a core component that tries to successfully address the needs of the new and always changing student populations (Kennedy, 1998). As mentioned by Kamat and Sardessai (2012), in any education system, the most important processes and areas of work are teaching/learning, evaluation and administration. As mentioned by Trevvett (2013), higher education institutions make significant investments, that range in the millions of dollars, while trying to successfully implement software systems in these areas. All of this is done without considering the internal cost of staff hours needed to make the project successful. Consequently, higher education institutions have found themselves in the need of multi-disciplinary software development teams to build software applications that adapt to their more particular needs. And for this, the identification or creation of a SDM that fulfills their software development needs with a single, unique approach results imperative.

Kennedy (1998) presented a view of the challenges software development teams in higher education could and would experience. They also did an analysis of different SDMs to identify the advantages and disadvantages those methodologies would present to teams developing software for this type of institutions. However, his recommendations were focused only on the analysis of teams dedicated solely to the creation of student learning software, leaving out all other software development teams that provide solutions for the remaining operations of the institutions they support, like assessment, research and administration.

Similar to this, Ibrahim and Oxley (2010) developed a methodology meant to be used by higher education institutions as well as libraries. However, this methodology targeted the development of mash-ups only, which are user applications that allow the combination of different sources of information, all of which can come from local or remote sources. Given that this methodology only addresses data aggregation needs, it cannot be considered a proper solution for more complex scenarios.

Other case studies, like Farrugia and Al-Jumeily (2012); Matijasevic, Roncevic, and Orel (2007); Pavolka, Mount, Neymeyr, and Rhodes (2005) presented stories where the use of certain SDMs for the creation of higher education software were successful. Farrugia and Al-Jumeily (2012) presented the case of a project meant to develop a web-based student-teacher's ePortfolio system for a university in Malta. The development group followed a rapid iterative process based on an Analysis-Design-Development-Implementation-Evaluation (ADDIE) model, resulting in an application described by the users as useful and easy to use and learn.

Similar to this, the study of Matijasevic et al. (2007) described the case of a project team that had to develop an information system to accommodate the demands of the the Bologna Declaration in Croatia, with the intend to make the educational programs of higher education institutions in Europe more unified and compatible. The project team decided to adopt an agile development methodology, resulting in a system delivered in three months and that has been widely adopted by the higher education community.

Finally, in his study, Pavolka et al. (2005) presented the case of a university that started the transition to a new enterprise-wide application that combined content management, electronic portfolios and collaboration and learning. Because of the different challenges experienced throughout the project by the team, they decided at some point to adopt a Rapid Collaborative Prototyping model, resulting in valuable user feedback to help to drive decision making.

Even though all three projects resulted in very successful systems, none of the articles addressed the need of having a single SDM that would properly mitigate all the challenges associated to the development of applications for higher education.

2.2 Software Development Methodologies

Although every software project is very different and unique (Dyck & Majchrzak, 2012), common patterns can be found among them. These patterns

have been identified throughout the years and put together in the form of SDMs. A SDM defines a set of best practices for the development of software, with a high level of abstraction (Magdaleno, Werner, & Mendes de Araujo, 2012). They can be viewed as recommended and proven practices to successfully achieve the development of a system throughout the whole life cycle of the project (Vavpotic & Bajec, 2009). SDMs provide support and structure to projects by describing the different processes through which each one of them should go through (Dyck & Majchrzak, 2012). They also help in the regulation and control of those processes during the development of systems (Hannan, 2011). These methodologies are often defined by different activities performed by the project team, and contain definition elements that go from phases and iterations, to roles, standards and expected results.

Independent of the size, type, complexity or industry, nowadays a good percentage of software projects are still failing, and only 39% of all projects are considered a success (Lynch, 2013). The constant need to improve the quality of the software (Magdaleno et al., 2012) and to avoid project failure has been the primary drivers for the creation and adoption of SDMs (Dyck & Majchrzak, 2012). The nature of the software industry, instead of simplifying, is getting more complex every day, and the development of large-scale, distributedly-developed systems adds even more complexity to the equation (Magdaleno et al., 2012).

Unfortunately, no single methodology has been developed to successfully address all this complexity or that can be easily adapted to all kinds of software projects (Glass, 2004). This is the reason why Vavpotic and Bajec (2009) indicated that many software development organizations do not use or have stopped using formal SDMs as part of their development life cycles altogether. They also explained that the risk of non-SDM adoption relates in the majority of the cases to two different aspects. In the first place, existing SDMs are not technically tailored to the specific organizational needs. And, in second place, existing SDMs do not fit the social features of the organization and their development teams.

The software development life cycle (SDLC), as defined by Hannan (2011), "is a structure imposed on the development of a software product" (p. 249). SDLCs are in most cases models created to follow systematic and disciplinary approaches in the creation of software solutions, meant to reduce the probability of chaos and failure of projects (Mahanti et al., 2012). Most SDMs do not cover all the phases of the SDLC and just a very few of the SDMs available cover all the development-related phases. Many SDMs tend to forget about the phases that come after development has finished, like operations, maintenance, enhancements and replacement. These, too, are part of the SDLC (Dyck & Majchrzak, 2012).

Many have attempted to develop a universal criterion that would allow software development teams to choose a SDM that better adapts to their needs, but all the different approaches have shown to have weaknesses, including limited scope, lack of transparency, and lack of detail (Dyck & Majchrzak, 2012). As pointed out by Mahanti et al. (2012), there has not been a model that could universally fit all development environment setups and that can be considered adequate in all situations. Thus, a SDM that adapts to the specific needs of an industry and that provides full coverage of all the SDLC, while at the same time faciliates the management of projects by the team or project manager is yet to be developed.

Vavpotic and Bajec (2009) tried to develop a framework for SDM evaluation that would allow teams to determine the best SDM for them to use. However, their scope of review was relatively small. Additionally, the evaluation was very extensive and extremely time consuming, making it almost impossible for organizations to use their approach in a timely manner to determine the most suitable SDM for their project needs.

Unfortunately, the challenges presented by Vavpotic and Bajec (2009) seem to be the norm. Hardly any practitioner is able nowadays to implement a SDM the way it is presented in its theoretical description, mostly because of the nature of the organization itself, the way they do business, and the different processes of software engineering (Dyck & Majchrzak, 2012). This enforces the idea that, instead of

trying to look for generalization in all industries, SDMs should be adapted and adopted to the convenience of software development teams, at least to a level that better fits their organizational needs. This adaptation, in some cases, may need the mix of two or more SDMs for it to properly fit to the needs of the organization.

2.2.1 Known Software Development Methodologies

Since the invention of software, any system that goes beyond trivial user needs will always evolve, even while still in development (Davis, Bersoff, & Comer, 1988). This becomes the main reason why many projects are behind schedule (as developers try to accommodate new requirements under the same time frame) and also the reason why systems fail to meet the expectations set by the customers (as developers may not acknowledge some of the changes and end up developing obsolete functionality). To address this, SDMs have been created. These methodologies provide a series of basic guidelines to develop software. These guidelines use engineering techniques, resulting in a sequence of stages and software evolution (Mahanti et al., 2012).

Davis et al. (1988) identified five as the most well-known SDMs used by the industry:

- The classical waterfall model, which is considered by many as the backbone of all SDMs (Mahanti et al., 2012). This methodology structures all phases of the SDLC as a cascade, where the output of a phase becomes the input of the next one, without allowing any process backtracking.

- Iterative waterfall model, which is an adaptation of the classical waterfall model but that allows a way to go back to preceding phases of the project to facilitate the correction of errors found. After detecting a problem, the team must fix the problem at the root, and then update all information for the subsequent phases.

- Rapid throwaway prototyping proposes an approach to ensure that software products will meet the users' needs by allowing quick and dirty implementations (or prototypes) of the functionality desired. Once users interact with the prototype and provide feedback, developers will then implement the real functionality to reflect the user needs.

- Incremental development is a methodology that encourages the partial implementation of a system and that allows for slow additions in functionality and performance in an incremental way.

- Evolutionary prototyping is pretty similar to rapid throwaway prototyping in and incremental development. It allows developers to construct a partial implementation of a system (mostly of well understood requirements) and then lets the users make use of this implementation to provide feedback and make sure the requirements were well understood.

- Automated software synthesis is a methodology where requirements are transformed from a high-level description into operational code by using knowledge or algorithmic techniques.

This list, although slightly different, presented many commonalities to a similar list explained by Kennedy (1998).

Mahanti et al. (2012), expanded the list of existing methodologies to include a few more:

- Code and Fix model, which describes the beginning of software development in general, where developers simply write code and then try to fix problems found. Because of its simplicity, this model is considered a two-phase methodology.

- V-shape model is similar to the classic waterfall model by providing a sequential path of execution of processes, although it differences by providing a stricter concentration in the testing of each phase.

- Unified process model, which is a use case-driven methodology, concentrated in the architecture of the application, using an iterative and incremental approach. This model consists of five phases: Inception, elaboration, construction, transition and production.

- Spiral development involves the repetition of the most common phases of the classic waterfall model, until the system is complete.

- Agile software development, which encompasses a set of guidelines and philosophies to encourage customer satisfaction, incremental software development, small and highly empowered teams, informal methods, minimal upfront planning and simplicity at its most.

When looking for a way to group this methodologies, they can be classified in many different ways. One of these categorizations is the provided by Ramsin and Paige (2008). They classify SDMs as fundamental methodologies (including Waterfall and Spiral Model), integrated methodologies (including Rational Unified Process and V-Model), and agile methodologies (including Scrum and Extreme Programming).

A more simplified approach is provided by Boehm and Turner (2003), on which software development methodologies are classified in only two different categories: plan-driven methods (which are considered the traditional way to develop software by establishing a well-defined process to follow during the SDLC), and agile methods (which consist of all the methods that focus in rapid prototyping and development, and that define the development of applications more as a craft than a process as seen in other industries).

Boehm and Turner (2003) proceed to expand into the differences between these two methods by the most common characteristics found on each. For plan-driven methods, the authors identify a very systematic engineering approach, on which the development of software has to carefully follow specific processes and phases until its full completion. A project is considered complete if not only the

code but also all its associated documentation is finished. Given its strong focus in documentation and process following, plan-driven methods depend heavily in a strong management of the process in order to be successful. These process must be constantly reviewed and analyzed in order to adapt them to the most current environmental circumstances.

In agile methods, the main approach is to provide a more flexible and adaptable environment for the development of software that better fits the rapidly changing nature of the software industry. These methods characterize as very usually being lightweight processes with short iterative cycles. The proponents of agile methods identified as the four major values of agility the interaction of individuals over process creation and tools, the development of code over comprehensive documentation, the focus in customer collaboration over contract negotiation, and the ability to respond to change over sticking to a predefined plan (Martin & Turner, 1986).

Based on all this, it is pretty clear that both the software development and the software engineering communities in general have no consensus not only on what methodologies exist and better adapt to the development of projects, but also on how the existing ones can be grouped and classified for better user understanding and selection.

2.3 Comparing Existing Software Development Methodologies

As presented in the previous sections, there is a wide range of known SDMs, as well as of different ways to classify them, in order to facilitate common user understanding. This makes the selection of a single methodology to be followed by software development teams a daunting task, becoming one of the many reasons why software development teams decide not use SDMs in real practice and, if they do, they do not follow them rigorously, as explained by Vavpotic and Bajec (2009). This is particularly true for those methodologies whose purpose is to be as generic

as possible, instead of concentrating in solving the needs of a particular organizational type or project, as well as the social characteristics and needs of the team themselves (Vavpotic & Bajec, 2009).

When trying to address the specific needs of higher education in the selection of a SDM that would adapt well to the development of teaching and learning software, Kennedy (1998) proceeded to compare four typical models regularly used in software development:

- Linear sequential model, better known as the waterfall model
- Prototyping Model
- Rapid Application Development, also known as RAD
- Evolutionary software process models, most commonly known as Incremental and/or Spiral model

As part of this comparison, the author analyzed the advantages and disadvantages of each model. Table 2.2 summarizes their findings.

Table 2.1.
Kennedy's SDM comparison

Model	Advantages	Disadvantages
Linear Sequential Model	Provides a template for all the different phases of the SDLC. It is widely used and well-known. It is better than not using a SDM at all.	Real projects are never linear. Any type of iteration causes confusion in the process. Stating all requirements upfront and correctly is very difficult. First versions of the project are delivered very late in the project time-span, so major problems are discovered too late. Even the smallest delays will affect the whole project schedule.
Prototyping Model	Fits well projects with fuzzy requirements. The prototype serves as requirement representation mechanism. Evaluation initiates early, so users can influence the final product.	Users get a wrong perception of how long it takes to develop software because of the prototype speed. Bad practices used in the prototype code may remain in the final software. Users get wrong expectations of what can be done with the allocated time and money When prototypes are finished, teams may lose enthusiasm.
Rapid Applications Development Model	A first version of the product can be demonstrable in a short period of time. Design and development is done incrementally. Users are involved early and throughout the project.	The model relies on re-usable components. Scope and requirements must be constrained for it to work. It works better for business applications. It may require multiple teams working concurrently in larger projects.

Table 2.2.
Kennedy's SDM comparison (continued)

Model	Advantages	Disadvantages
The Incremental Model	Uses the linear organizational components of the linear sequential model with the iterative approach of the prototyping model. It can produce a usable product quickly, although with limited functionality. Each iteration delivers an operational product. Feedback from users and usability testing comes early so product can be adapted. User involvement starts early in the process. Increments can be planned ahead easily.	It is difficult to pre-empt what functionality will be needed in the future. Integration of different components can be difficult. The scope and requirements of the project must be constrained for the project to succeed. The model works better for systems that can be delivered as a series of interoperable components.
Spiral Model	Uses the iterative approach of the prototyping model with the technical and systematic components of the linear sequential model. Allows rapid development and prototypes that can then be evaluated by the users. Users can provide early feedback in the project. Supports long term projects that will require a lot of changes and adaptation. It is a better representation of the life cycle of a project	It is not as recognized as some of the other models. Lack of detail and process may cause a bad perception among stakeholders. Risk assessment is vital early in the project. The success of the project relies heavily in the project manager's knowledge and expertise.

Although this analysis provided a good perspective on the advantages and disadvantages software development teams developing teaching and learning technologies for higher education would face while following one of the methodologies previously mentioned, the author fails to address how well these methodologies would fit teams developing software for other key areas, like administration and research. Hence, for the purpose of this study and in order to provide a more holistic review of SDMs that could be used by higher education teams, there is a need to supplement Kennedy (1998)'s analysis with one that provides a more generalized review of SDMs in general.

In an intent to explain SDMs from a more simplistic but yet philosophical perspective, Boehm and Turner (2003) categorized all SDMs in only two different categories: plan or discipline-driven methodologies and agile-driven methodologies (both briefly described in the literature review chapter). In order to compare these two categories, the authors identified four project characteristics to be considered, those being:

- The application characteristics, which include the primary project goals, its size, as well as the application environment
- The management characteristics, which include customer relations, project planning and control and project communications
- The technical characteristics, referring to techniques used for requirement definition, development and testing
- The personnel characteristics (which include the customer and developer characteristics, as well as the organizational culture)

The summary of the comparison presented by Boehm and Turner (2003) can be found in Table 2.3.

Boehm and Turner (2003) did not define the set of characteristics used in this comparison with any specific industry in mind. Because of this and the fact that their approach also encapsulates all the different methodologies discussed by Kennedy (1998) in their review, their comparison could be used as a good supplement to provide more complete selection criteria for software development teams working for higher education to decide what SDM works better for their needs. This is especially true for those teams developing software for areas that were not specifically addressed by Kennedy (1998) in their study.

2.4 Summary

This chapter provided a review of the literature that supports the main argument of this research project. In first place, an outline on how software development relates to higher education was presented, as well as an overview of existing literature related to this area. This was then followed by an examination of SDMs, and a quick introduction to the most well-known SDMs was given. The chapter ends with the comparison of comparison of existing SDMs, utilizing two previous studies done around this topic.

In the next chapter, the author proceeds to explain the theoretical and methodological framework used in this research project.

Table 2.3.
Boehm and Tuner's SDMs comparison

Type	Characteristics	Agile-Driven	Plan-Driven
Application	Primary Goals	Rapid value. Responding to change.	Predictability, stability and high assurance.
	Size	Smaller teams and projects.	Large teams and projects.
	Environment	Turbulent. High change. Project-focused.	Stable. Low-change. Project/organization focused.
Management	Customer Relations	Dedicated on-site customers. Focused on prioritized increments.	As-needed customer interactions. Focused on contract provisions.
	Planning and Control	Internalized plans. Qualitative control.	Documented plans. Quantitative control.
	Communication	Tacit interpersonal knowledge.	Explicit documented knowledge.
Technical	Requirements	Prioritized informal stories and test cases. Undergoing unforeseeable change.	Formalized project, capability, interface and quality. Foreseeable evolution requirements.
	Development	Simple design. Short increments. Refactoring assumed inexpensive.	Extensive design. Longer increments. Refactoring assumed expensive.
	Testing	Executable test cases define requirements.	Documented test plans and procedures.
Personnel	Customers	Dedicated and collocated.	Not always collocated.
	Developers	Tends to need a richer mix of higher-skilled people.	Are able to operate with less-capable people. Still needs a few highly skilled resources.
	Culture	Comfort and empowerment via many degrees of freedom.	Comfort and empowerment via framework of policies and procedures.

CHAPTER 3. THEORETICAL AND METHODOLOGICAL FRAMEWORKS

Relevant to this study is a discussion on the concept of Activity Theory (AT) as a theoretical framework. This approach studies the actions of people using activities as the units of analysis and how this framework can be adapted to define software development activities. First in this chapter, the author will provide a quick review of AT, followed by an analysis on how this theoretical framework has been utilized so far in the area of software development.

Following, the chapter continues with a review of Case Study Research (CSR) as this study's methodological framework. CSR is a type of qualitative research focused in the investigation of phenomena happening in real life, by providing a description, understanding, predictability and control of the different entities under review. As the research methodology to be used in this study, the author explains how CSR, compared to other qualitative research approaches, is the most appropriate one to address the research questions of this study.

3.1 Activity Theory

As described by Levy (2008), "activities are the center of human behavior" (p. 1664). They are understood as the relationship between a subject and an object that transform both entities (Kaptelinin & Nardi, 2012). A subject, described in a simplistic way, is an agent that undertakes an activity, trying to reach an outcome (Barthelmess & Anderson, 2002). This outcome requires the transformation of an object. An object could be from something material, to something less tangible or totally intangible. The main characteristic an object must have is to be sharable for

manipulation and transformation by the subjects involved in an activity (Kuutti, 1996). Communities, which are groups of subjects, usually share an object and collaboratively work together to transform it. This process as a whole can be considered an activity.

Activities, in a narrow sense, are units of subject and object interaction defined by a motive. They are considered a system of processes oriented towards a motive where the meaning of any individual component of the system is determined by its role in attaining the motive (Kaptelinin & Nardi, 2012). Activities, as defined by AT, provide enough contextual information to make an analysis meaningful, while avoiding a narrow focus on an individual or too broad a focus on whole social systems (Barthelmess Anderson, 2002). Fjeld et al. (2002) outlined two types of activities: goal-direction and goal-derived activities. In goal-directed activities, actions are derived from a goal setting, while in a goal-derived activity, the goal settings are derived from the actions.

Human activities are the primary concept in Activity Theory (AT) (Barthelmess & Anderson, 2002). AT can be defined as a philosophical and cross disciplinary framework (not a theory as its own name might imply) for studying different forms of human practices as development processes, with both individual and social levels interlinked at the same time (Kuutti, 1996). It is a social theory that describes human consciousness as the product of the interaction of a subject with other people and objects, all this while using artifacts to facilitate everyday activities (Kaptelinin & Nardi, 2006). Its core idea is that, by analyzing the context of outcome-driven activities, a full understanding of the actions and operations individuals take to reach such outcome will be obtained (Döweling, Schmidt, & Göb, 2012).

Central to the concept of activity theory, is the concept of mediation (Barthelmess & Anderson, 2002). The shaping of all human experience is defined by the tools and sign systems we use (Nardi, 1996). The role of mediators is to connect humans to the world, in an organic and intimate way. The relationships between a subject, an object and their community are mostly mediated by the instruments used, the rules imposed and the agreed division of labor. These mediators are used by the subject and the community to achieve a desired set of transformations to an object. This mediation is better exemplified in the structure of a human activity system diagram provided by Engeström (2015) and shown in Figure 3.1.

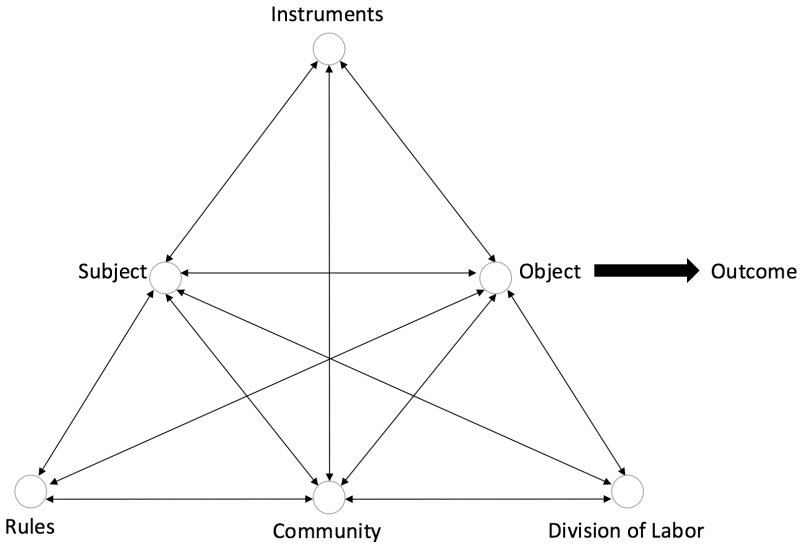

Figure 3.1. The structure of a human activity system

The mediating artifacts between subjects and objects are not static entities. They are constantly revised and transformed to better meet the always changing needs of the community by embodying their collective experience (Bardram, 1997). In this mediation, community refers to virtually all of the people directly involved in the particular activity being analyzed, while instruments are the entities that shape the way that people interact with reality. Along with this, rules are the

domain-specific knowledge that must be captured somehow by the subjects, while the division of labor refers to the cooperation and specialization occurring in an activity (O'Leary, 2010).

The relationship drawn between the elements described above reflects a set of activities, actions and operations undertaken by a subject while producing an outcome. Activities are the highest order frame for objectives, while actions and operations designate lower level acts embedded in activities. Subjects rely on instruments to help reach objects, while instruments help to mediate activities between subjects and objects (Tan & Melles, 2010). Activity theory thus allows for a rich description of an activity system utilizing these terms, allowing to describe activities as products of the intentions of acting human agents (Nardi, 1997). This description is achieved by visualizing the various levels of activity in professional practice situations, firstly by understanding how subjects utilize tools, and secondly by focusing on the social dynamics of subjects in context (Redmiles, 2002).

Being AT a research approach and framework that can adapt to multiple disciplines (Barthelmess & Anderson, 2002), it has been commonly utilized for the study of activities of work and technology (Engestrom, 2000).

3.1.1 Activity Theory in Software Development

As described before, AT is a framework to analyze human activities based on the interactions between subjects with objects and other subjects, while using tools and following rules. A way to exemplify these interactions is the development of a software application by a team. In software development, multiple subjects (software developers, team leaders, project managers and, in some cases, customers) work collectively (during the project lifecycle) to reach the transformation of an intangible object, which is the software itself.

AT is particularly useful in the analysis and understanding of collaborative contexts, independent of the field. Given the complexity of current software applications, software development is, in most cases, a highly collaborative environment. This argument is used by Barthelmess and Anderson (2002) to analyze software development from this perspective. Döweling et al. (2012) acknowledged that humans by nature, and usually in collaborative environments, try to reach their goals and objectives by organizing their work in tasks or activities, which supports the reasoning behind the use of AT to shape software development as a collaborative activity.

AT also helps to highlight the context perspectives of a process-oriented environment, which differs significantly from product-oriented approaches commonly found in software engineering (Barthelmess & Anderson, 2002). In software development, a process-centered environment is mostly concentrated in developing the right processes used to produce and maintain systems, more than in the product to be developed. However, no literature was found that provided a good model to allow organizations to shape process-oriented software development environments to their specific needs using AT.

Hannan (2011) explained this a little further by pointing out that an activity in AT can be represented as a big task (e.g., the development of a software application) or as a small task (the implementation of a specific functionality of the application). The author considered this one of the biggest limitations of AT, as it does not demonstrate consensus among researchers on the correct definition and scope of the activity term within an AT content. Barthelmess and Anderson (2002) disagreed with this perspective. They believed one of the biggest advantages of AT in the study of software development methodologies is its lack of a strict definition of what an activity is. By avoiding to narrow the focus of the analysis on just

individuals and by broadening the focus to social groups, AT provides enough contextual information to make meaningful the analysis of any methodology, including those for software development.

3.2 Case Study Research

As defined by Yin (2009), case study research (CSR) "is an empirical inquiry that investigates a contemporary phenomenon within its real life context, especially when the boundaries between phenomenon and context are not clearly evident" (p.13). Woodside (2010) proceeded to expand on this definition by indicating that this type of inquiry focuses not only on the investigation of the contemporary phenomena but also in the description, understanding, predictability and control of the individuals under review, which is the main and final objective of CSR. This type of research strategy can involve either single or multiple cases and can also employ multiple levels of review under the same study, with the main purpose of understanding the dynamics between the different settings (Eisenhardt, 1989).

CSR, as any other empirical research, has a research design. A research design is the sequence of steps to be followed to connect the data collected to the research questions and draw a conclusions based on the results (Yin, 2009). The research design for CSR contains five especially important components: (a) the study questions, (b) the study propositions, (c) the units of analysis, (d) the logic to link the data collected to the propositions, and (e) the criteria used to interpret the findings (Yin, 2009).

As any other research methodology, CSR faces several criticisms, being the most important ones: (a) it fails to confirm the independency of the results obtained from the only case studied, and (b) it fails to gain deep understanding of the mechanics involved in the process under study (Woodside, 2010). Researchers using

CSR as their research approach can counter these criticisms by avoiding generalization of conclusions and results.

CSR typically combines different data collection methods to create saturation and validate the evidence found (Eisenhardt, 1989) and it is mostly used to provide description of a phenomenon, test an existing theory or develop theory by itself. For the generation of theory, Eisenhardt (1989) defined a process that can be summarized as a sequence of the following steps:

- Definition of research questions, which allows the researcher to look for the specific data relevant to the study.
- Selection of cases, to identify the set of entities from which the research findings will be drawn.
- Craft of data collection instruments and protocols, to provide a stronger substantiation of data to support the research constructs and hypotheses by providing multiple data entries.
- Data collection and analysis overlap, to provide the researcher an early start in the data analysis while still gathering information, as well as to allow the researcher to take advantage of data collection flexibility while still on the field, based on the preliminary analysis.
- Case analysis, which allows the research to start looking for the answers to the research questions by identifying unique patterns that emerge within the data collected, giving the research a deep familiarity with the case under study.
- Hypothesis shaping, which allows the researcher to compare the theory with the data collected, in order to start creating the new form of concepts and valid theory.

- Enfolding of literature, to compare the new concepts and theory with existing ones, in order to examine possible conflicts and contradictions that could jeopardize the validity of the study.
- Research conclusion, which summarizes the process and brings the study to closure, once data and case saturation have been reached and no further information can be drawn.

This research project has concentrated in the creation of a theory for a particular research context, more specifically, in the development of a SDM for teams working in higher education. This, along with the connection between the reality of the environment, subject's experience and existent research as the main drivers of this approach, clearly makes CSR the qualitative research approach of preference for this study, particularly when applying the process suggested by Eisenhardt (1989) for this specific purpose.

3.3 Summary

This chapter described AT as a theoretical framework that allows the analysis of the interactions between subjects and objects, and how those interactions can be affected by the community surrounding them, as well as by the twenty two artifacts used and the rules imposed to those interactions. These section also explained the value AT can provide to the development of new SDMs, given the nature of software itself and the interactions needed between the different actors involved in the development of applications. The chapter closed with a review of CSR, and the reasoning behind selecting this approach as the most appropriate one to address the research questions of this study.

In the next chapter, the author proceeds to explain the methodology used in this research project.

CHAPTER 4. METHODS

This chapter describes the different procedures used in this research project to answer the questions: How are software development methodologies currently being used by software development teams working for higher education institutions? How are these methodologies supporting or limiting the team's performance and outcomes? What is the most effective way for these teams to develop software in order to minimize the limitations most commonly found, while at the same time leveraging some of the advantages provided by existing software development methodologies?

As one of the goals of this research project was to understand how the peculiarities of a higher education setup can affect software development teams at Purdue University, a qualitative approach was determined to the best method to collect the data needed to address the goal established. This chapter explains how data was collected for that purpose and how the results obtained were analyzed to build the case and accomplish the goals set.

4.1 Case Study

As Eisenhardt (1989) indicated, organizational research revolves around the development of theory, and this is accomplished by combining existing literature, common sense and people's expertise. The intimate connection between the data collected during this type of research and its empirical reality is what allows to develop valid theory that is relevant and testable. As this research project was of organizational research nature and tried to determine the best way to design a

software development methodology that addressed the most common challenges teams faced while developing software for higher education institutions, it was determined that a case study qualitative research approach was the most appropriate methodology to accomplish this goal.

For the case study, data collection was accomplished in the form of interviews to current software development team managers and developers at Purdue University. Information about the advantages and challenges these people and their teams faced on their software development life cycles (SLDCs) while developing applications for the institution was gathered, as well as information about the methodologies these teams used to accomplish their projects.

After the collection of the data, the information gathered was analyzed using a qualitative methodology known as thematic analysis. At the same time and based on existing literature, a review of well-known software development methodologies was done, where advantages and disadvantages were determined for each, resulting in an instrument that was later used by the author to analyze the themes identified from the data obtained during the interviews. This analysis helped to determine what components of the methodologies reviewed could better address the challenges found in the different themes.

With the results obtained from this analysis and with the help of AT, the author then developed a new SDM. This methodology addressed for the most part the challenges most commonly identified during the interviews. It incorporated some advantages of existing software development methodologies and was designed in such a way to facilitate easy adoption by software development teams in higher education.

With this approach, all the steps in the process of building theory from case study research described by Eisenhardt (1989) were met: definition of research

questions, selection of cases, craft of data collection methods, overlap of data collection and analysis, case analysis, shape of hypothesis, enfolding of literature and conclusion.

4.2 Participants

Homogenous sampling was defined as the method to be used for the selection of the team leaders, managers and developers that were invited to participate in the interview phase. The idea behind the homogenous sampling method is to describe a particular subgroup in depth (Patton, 2002). As most managers and leaders in charge of software development teams at Purdue are usually in charge of the selection of the software development methodologies used by their teams, they were considered to be the subgroup from which more in depth information should be obtained. Even though each team is considerably different from all the other ones, they all have the common purpose of serving the software development needs of different areas at Purdue. Thus, this sampling strategy was considered to be the most adequate one.

4.3 Data Collection Methods

Interviewing can be defined as a conversation that has a specific purpose and this purpose is to gather information of interest (Berg, 2009). Given that Purdue University has multiple software development teams on campus, purposeful conversations with the leaders and members of those teams provided an insight on what are the biggest challenges they face while developing software for the institution, what steps have they taken to address those challenges, and if, how they have adopted software development methodologies to overcome those as well.

Given the constraints in time and resources, the number of interviews was narrowed to ten. The interviews targeted teams that support different areas at the university, including two academic areas, two administrative areas and one research-related area.

All interviews were semi-standardized, meaning that even though a predetermined set of questions had been selected and each question was asked to each interviewee in a systematic and consistent order, the interviewer was allowed some freedom to digress and deviate from the interview protocol to probe far beyond the answers received from the interviewees (Berg, 2009).

4.3.1 Interview Protocol

A first version of the interview protocol was prepared and presented to three subject matter experts for their review and correction. All three reviewers, which included Professor Jeffrey L. Whitten, Professor Kevin C. Dittman and Professor Jeffrey L. Brewer, full time professors of the College of Technology at Purdue University, who provided meaningful suggestions and feedback to the interview protocol. The resulting protocol can be found in Table 4.1.

As for the columns of the protocol, RI stands for relevant information, PM stands for project management, AT for activity theory and SDM for software development methodology.

4.4 Procedures

A one-hour interview was scheduled with each one of the selected developers and team leads, managers or directors during the month of August, 2016. The interview protocol presented in the previous section was used on each interview and

Table 4.1.
Final Interview Protocol

	Question	RI	PM	AT	SDM
	Please provide a description of your job duties	✓			
	How many years of experience do you have in application/software development as a manager and/or developer?	✓			
	Could you please describe your team in terms of: • Team Size • Skills • Location (Collocated, Remote, Distributed) • Staff Mix (Full Time, Temporary, Consultant, Contractor) • Average Development Efforts (0-3 months, 3-6 months, 6-12 months, 12+ months)	✓		✓	
	Are software applications developed by your team managed as projects? If not, how are they managed?		✓		
	Could you please provide a list of the different stakeholders and groups that get involved in your software projects? Please remember to include stakeholders as other IT peer groups (security, customer relations, infrastructure) as well as other university entities (internal audit, HR, business services)		✓	✓	
	What are the most common challenges and the biggest problems your team faces while developing applications for Purdue University?		✓	✓	
	How do any of the stakeholders you mentioned contribute to the most common challenges and biggest problems that you mentioned above?		✓	✓	
	Does your team follow any software development methodologies while working on your software development efforts?		✓		✓
If Yes	Could you please provide a detailed description of the different activities that take place during a normal project that would follow such methodology?		✓	✓	✓
	Have you changed this methodology from its theoretical description to better adapt to the needs of your team?		✓		✓
	What advantages have you found in this methodology?				✓
	What disadvantages have you found in this methodology?				✓
If No	Could you please explain why a methodology is not being followed?		✓		✓
	How willing would you be to adopt a software development methodology that has been specially tailored to software development projects in higher education?		✓		✓

each interviewee accepted the interview to be voice recorded. After each interview, the audio was transcribed by the author and stored in a digital format to be used for future reference. All recordings were properly discarded after each transcription.

4.5 Data Analysis Methods

Thematic analysis with a post-positivist approach was the method used for data analysis and interpretation of the data collected during the interviews. Thematic analysis is a process for encoding qualitative information (Boyatzis, 1998) that helps to identify, analyze and report themes (or patterns) within the data (Braun & Clarke, 2006). A post-positivist approach allows researchers to look for patterns in human actions and behaviors.

As one of the goals of this research project was to understand what are the biggest challenges of the software development methodologies currently in use by software development teams working for Purdue University, it was considered that with the use of thematic content analysis, different themes or patterns could be identified from the raw transcript data obtained from the interviews. This allowed the author to identify the most common challenges the software development teams at Purdue University faced, and how the teams have tried to address these challenges, with or without the use of software development methodologies. As this research looked to understand the behaviors of the different interviewees and their teams under the given circumstances, a post-positivist approach was the best fit for the needs of the project.

Once the themes were identified, they were validated against an instrument developed by the author using existing literature on well-known software development methodologies. In this review, advantages and disadvantages of each methodology were determined and compiled to determine the assessment criteria.

This instrument helped the author review the different themes found on the interview data and determine what components of the methodologies reviewed better addressed those themes.

With the results obtained in this analysis and with the help of AT, a new software development methodology to be used by teams developing solutions for higher education institutions was developed. That is, elements of AT were mapped to specific findings and the components of the new methodology were explained through an AT lens in the data analysis chapter.

4.6 Summary

This chapter provided a review of the methods used in the research study. The author explained the selection of case study as the approach to reach the research goals and then proceeded to expand on how participants for the study were selected, what procedures were used, how data was collected and how it was also analyzed to draw the final conclusions and results.

In the next chapter, the author proceeds to explain how the data obtained for this study was analyzed and then proceeds to explain the logic behind the resulting methodology proposed in the study, and the different steps taken to define it.

CHAPTER 5. DATA ANALYSIS

After the completion of the interviews scheduled with all ten participants that participated in the study, the author proceeded analyze the results obtained. This chapter provides a summary of the different steps taken by the author to do the data analysis and provides some arguments to confirm the validity and reliability of the study.

5.1 Data Processing

First, the author proceeded to transcribe the voice-recordings of each interview into a text document. Once all transcriptions were completed, the author proceeded to properly discard the voice recordings to protect the participants' privacy and also to reduce bias in later phases of the analysis, given that no interview could be easily traced to a specific person, based on the deep level of aggregation the data analysis process reached.

5.2 Themes Identification

Once all the transcriptions were ready, the author then proceeded to read the interviews one by one, and started to identify and highlight the different topics or themes that were mentioned and discussed by each participant. After this initial review, the author then proceeded to review the text of each interview again, and started to breakdown the data into concepts and categories, based on the different topics found during the interviews. To do this, the author first gave each interview a

unique numeric identifier that could be used as a reference in the future, in case a review of the original text of the interview was necessary. After this code assignment, the author then proceeded to compile the different concepts or themes found in the interviews following an open coding structure. An example of this structure can be found in Table 5.1.

Table 5.1.
First level concepts

Open Code	1	2	3	4	5	6	7	8	9	10
Stakeholders do not understand the scope of their project			P7		P3					P5
Continuous improvement of existing applications is fairly common	P4	P6					P11		P6	
Following a methodology only for standard to complex projects		P12		P3		P9		P2		

5.3 Compilation and Coding

As part of the compilation and coding phase, the author reviewed the text of each interview again and started to track each concept found in the conversations as a new row in the coding structure. Following, the author recorded the page number where the concept was mentioned in the interview under the column that identified the specific interview where it came from (demonstrated in the example with the numbers one to ten). If the concept was mentioned multiple times during the same interview, the author proceeded to track each page where it was mentioned. If the concept had already been mentioned in a different interview and was already being tracked in the coding structure, the author only added the page number where it was referenced under the corresponding column for the given interview.

Once this code compilation phase was completed, the author proceeded to count the number of times each concept was mentioned in different interviews, to help identify how commonly was the given theme recognized by the participants. After this, the author proceeded to classify each of these concepts into different categories using axial coding. A short example of this exercise can be found in Table 5.2.

Table 5.2.
Axial coding

Axial Code	Open Code	Count
Biggest challenges	Customer indecision	3
Development efforts	Continuous improvement of existing applications is fairly common	4
Methodologies	Following a methodology only for standard to complex projects	4
Process	Requirements gathering	6
Tools	Project management tool for tracking	1
Current Process Advantages	Flexibility	3
Current Process Disadvantages	Process is not rigid, it lacks formality	2

A total of 180 themes were identified out of the different interviews and were classified in seven different axial codes.

5.4 Credibility

As mentioned by Patton (2002), validity and reliability are the two measurement instruments of which the quality of qualitative research depends on. The majority of the Methods and Data Analysis chapters were dedicated to fundament the different decisions made by the author around those two vital areas of this research study. The decisions there were sustained in the selection of well

respected and recognized methods and data collection techniques. However, the author is yet to address the steps taken by the author in this study in order to secure its credibility and demonstrate its validity and reliability. Such steps are explained in the following sub sections.

5.4.1 Credibility of Researcher

With over twelve years of experience in the development of software, of which eight of those years had been managing software development teams and five were working for a higher education institution, the author of this study fit what was described by Eisner (1991) as connoisseurship, which is one who has extensive knowledge about a particular domain of interest. In the case of this study those areas were software development and its particular application to higher education.

Although it is left up to the reader to decide if the author's experience and background was relevant to the study, the author completed this research project with the hope that both the combination of experience and well recognized methods and data collection and analysis techniques provided enough of an argument to sustain its credibility.

5.4.2 Intra-Rater Reliability

In order to verify the reliability of the information obtained from the interviews, the author proceeded to do the codification of the data and compilation of themes twice. The purpose of this exercise was to validate that the codification and theme identification had been done properly and no key information was left out during the first review of the data collected in the interviews. After the second round of codification and theme compilation was done, the author proceeded to

compare the results of each exercise and adjusted the themes with any information that was missed during the first round of data analysis.

5.5 Summary

In this chapter the author proceeded to review the different data analysis techniques used in this study, in order to obtain meaningful information out of the data collected in previous phases. The author also proceeded to provide different arguments in order to validate the credibility of this qualitative study.

In the following chapter, the author explains how the themes found during the analysis of the data were utilized to obtain the results that would fundament the final proposal made for study.

CHAPTER 6. RESULTS

In this chapter, the author provides an overview of the results found in the review and analysis of the data obtained during the interviews done for this study. These results will then be tied in the next chapter with the review of SDMs presented in the literature review chapter, in order to define an instrument that will later be used to make a final recommendation on the SDM teams developing software for higher education should follow to increase their chances of success.

6.1 Resulting Categories

With the information obtained during data analysis, the author was able to identify seven main categorizes based on the information provided by the different interviewees:

- Team demographic data
- Most common stakeholders
- Methodologies currently used
- Current processes
- Advantages of the current process
- Disadvantages of the current process
- Biggest challenges faced

Each one of these categories and themes captured data that served in one way or the other to develop the final recommendation of this study. Next is a description of each category found, as well as the information expected to be obtained from the data.

6.1.1 Team Demographic Data

From the demographic data obtained in this study, the author was able to identify that, in average, teams developing software for Purdue University have around ten team members, not counting their lead, manager and/or director.

Nine out of the ten interviewees indicated that their teams were all collocated in the same building to facilitate collaboration, meaning that distributed teams were very rare. As one of the interviewees indicated: "We are all in the same room, we have an open environment. Everybody can talk to each other, which is good and bad". Also, although offered as an option, telecommuting was seen by at least half of the interviewees as something their team members could only use under special circumstances, like extreme weather conditions or household needs.

A vast majority of the teams (n=9) concentrated their efforts in the development of web applications and defined their efforts as projects, as also indicated by the interviewees. These projects were usually never longer than six months, and a third of the interviewees indicated that the majority of their projects would usually take three months or less.

With this information, the author was able to get an idea of the average size of teams developing software for higher education, as well as the the size of the projects they were working on. This information was a key component in the design of the selection instrument created to determine the most suitable parts and pieces of existing SDMs, that would better fit the team's need.

6.1.2 Most Common Stakeholders

Given that this study tried to have a good representation of software development teams that served the most important areas of the university, including teaching and learning, research and administration, it was expected the range of stakeholders to be mentioned by the interviewees to be very broad, and that was exactly the case. When the participants were asked to provide a list of their most common stakeholders, these were the most commonly mentioned:

- Faculty
- Academic staff
- Students
- Administrative staff (including high ranks)
- Extension offices

When the interviewees were asked to expand this list with stakeholders from other peer groups that are also commonly involved in their projects, all of them mentioned other IT groups on campus, like Infrastructure Services and Security and Compliance, and seven out of the ten mentioned some of the vendors they have to work with on a regular basis.

With this information, the author got a fair understanding of the different parties that get to participate in the development cycle of software applications for higher education and, with the help AT, was able to define the most proper development methodology that would help better facilitate the interactions between the different people involved.

6.1.3 Methodologies Currently Used

When the author requested the participants to indicate if their teams used any particular SDM at the moment of the interview, eight of them indicated that their groups did, but followed this methodology loosely. As indicated my one of the interviewees: "we use the agile methodology loosely. It is not a hit and fast but thats what we use for the most part". Only one participant indicated that their group did not follow any particular methodology, although they had a good understanding of the existing ones, and only one participant indicated that their group followed a methodology exactly as it was theoretically defined.

Interviewees indicated that for the methodologies they used, in many cases they had to adapt it to better fit their group's needs. As said by one of the participants: "we have been gradually adapting and maturing our methodology based on what has been working". Some of the methodologies that were most commonly mentioned were waterfall and agile methodologies. The participants also indicated that, even though they followed some of the principles behind these methodologies, they were not necessarily following them on every step of the SDLC.

Half of the participants also indicated that role definition was fairly common in their projects, being the role of Project Manager one of the most prominent ones, especially for complex projects. Other roles mentioned throughout the interviews included architects, leads, quality engineers and developers.

With this information the author was able to understand how familiar teams developing software for higher education were with the concept of SDMs, with the different SDMs available for use, and to which extent these teams were currently using SDMs as part of their daily activities. This information also allowed the author to get a perspective on how open would teams in higher education be to the possible adoption of an SDM targeted to their own needs.

6.1.4 Current Processes

When the interviewees were asked to explain the process that they followed to complete their assigned projects, their answers gave a clear picture of the disparity between each team's practices. Although a good majority of them described the different phases of the SDLC as being part of their regular activities to complete a project, the biggest differences were noticed in the level of formality used by each group, especially during the first phases of the project (including project review, initialization and analysis) and the last few phases (testing and maintenance).

Elements of linear development methodologies (including minimum to none customer involvement during the development phase of the project and testing being executed as a separate phase in the development cycle) as well as of agile development methodologies (including biweekly delivery cycles and strong customer involvement during all phases of the projects) were commonly found in many of the processes mentioned.

Also, teams with extra levels of formality usually put more time and effort into the review, initialization and analysis phases, while interviewees working for teams with less formalized processes usually started explaining the steps followed to complete their projects from the project analysis and design phases. Design was the only phase of the SDLC that most interviewees acknowledge their teams did in a consistent way among all their different projects.

Nine of the ten interviewees mentioned testing as one of their steps during their development cycle, although exactly when this testing happened varied widely among groups. For example, two of the interviewees indicated that their teams started their testing during the project development phase, while the other seven indicated that they did it just after the all requirements were fully implemented.

Finally, only two of the participants included the support of the systems delivered as one of the phases of their processes. This could be possibly justified by the fact that over half of interviewees indicated that most of their projects went through several iterations. This meant that once the project was completed, it was very likely the team would have to work on a new version of the system in the future, reason why they did not consider support as part of the process, but a new project by itself.

This information was a key component for the author to understand that there does not seem to be a standardized way of developing software between teams at Purdue University, or even between the steps each one of the teams followed to complete their projects. This analysis also allowed the author to identify the phases of the projects that were considered to be the most important ones by the different teams, as well as those phases that each team considers vital for the success of their projects.

6.1.5 Advantages of the Current Process

After going through an overview of the process followed by their teams while working on software projects, the interviewees were asked to identify the biggest advantages they saw in following such processes. The responses varied widely between participants, making it difficult to identify common themes among them, but both having the ability to react to quickly to change and unknowns, as well as the ability to narrow the project scope to what is really needed were the two themes that got the most number of mentions, with over fifty-four percent of the participants indicating these to be advantages. As one of the participants mentioned: “The biggest advantages we have found is in our responsiveness”.

Six out of the ten interviewees indicated that for them to be successful in their jobs, their processes and methodologies had to allow them to be flexible, adaptable and to be able to react quickly to unknowns. This also includes unforeseen extra time needed to solve a particular problem, as well as time they would have to spend providing fixes for systems that were currently in maintenance.

Also, three of the participants considered as an advantage that, depending on each project, the teams could adapt the level of formality needed for each project, and cut unnecessary documentation if considered appropriate for the project's success.

6.1.6 Disadvantages of the Current Process

Same as with the advantages, the interviewees were asked to comment on the disadvantages they most commonly experienced with the processes their teams followed while working in software projects. The results obtained were as scattered as the ones obtained for the advantages, but with one very interesting characteristic: interviewees who indicated that their teams followed less formalized processes for the development of systems indicated this lack of formality as one of their major disadvantages; and interviewees who indicated that their teams followed very formalized processes indicated this much formality to be a disadvantage as well.

The lack of formalized quality assurance was another disadvantage recognized by a third of the participants. Although considered one of the most important phases of the SDLC, next to the development of the systems themselves, the interviewees recognized that their processes allocated minimum to no time to assure the quality of the systems, and that institutional support to have resources dedicated to these tasks had not been received at that point.

Some other minor but yet still important disadvantages acknowledged by the interviewees included scope creep and the lack of a supporting structure. With scope creep, the participants indicated that giving customers the opportunity to revisit and change the requirements of a project at any point throughout the development cycle could result in major changes to already existing functionality, causing significant delays in the final delivery of the systems. With the lack of a supporting structure, the interviewees indicated that when a project gets completed, they immediately have to start working on new assignments while, at the same time, having to deal with the support needs of the recently released systems. Many times project deadlines and estimations do not take in consideration the effort main resources have to invest supporting existing applications, and this constantly has an impact on their ability to deliver systems within the given deadlines.

6.1.7 Biggest Challenges Faced

In order to understand what are the limiting factors software development teams in higher education have to face in order to successfully complete their projects, we asked the interviewees to comment on the most common challenges they face while developing their systems. We also asked them to reflect on those challenges, keeping in mind the interactions they have with the different stakeholders that were usually involved in their projects.

When asked this question, the interviewees shared in length their thoughts, allowing the author to collect the majority of the data for this study in this particular section of the interview. After identifying all the different challenges mentioned by the interviewees and categorizing them as such, the author then proceeded to classify the different themes found in this category into different subcategories. For this, the author reviewed each gathered concept again and

proceeded to determine a more intrinsic meaning behind it, resulting in a subcategorization of the theme. Once all themes were reclassified into these subcategories, the author proceeded to aggregate the data even further by counting the number of times each subcategory was mentioned. The results of this exercise can be found in Table 6.1.

Table 6.1.
Biggest challenges found

Challenge	Times Mentioned
Customer interaction and lack of engagement	23
Resource constraints	13
Poor planning	12
Organizational issues	11
Collaboration with other groups	8
Scope definition	5
Process deficiencies	4
Lack of supporting structure	4
Poor change management	3
Lack of proper communication	3
Poor prioritization	3
Unclear governance	2
Vendor constraints	2
Fierce competition	2
Lack of influence	1

From this table the author could clearly identify what type of challenges were considered by the interviewees as the most important ones, given the number times they were mentioned. Customer interaction was considered the most important challenge of all. As expressed by half of the interviewees, customers tended to have a poor understanding of their projects and what they were really looking out of them, as well as of the overall impact their requests could possibly have for the university as a whole. Teams usually struggled trying to understand if the requirements given by the customers were a good representation of what they

really want, especially when customers kept changing the requirements over and over again. Customers also tended to request help from development teams by providing what they considered would be the best way to solve a specific problem, but when asked, they were not able to articulate the problem itself. As expressed by one of the participants: "they will often come to us with a proposed solution, instead of coming and defining a problem". A lack of understanding of the scope of the project, its impact, and the different parties involved also affected on a regular basis the capacity of software development teams to deliver a product that satisfies the customer's needs.

One particular area of customer interaction that was commonly mentioned among the interviewees was the lack of engagement from the customers soon after their projects got approved. Customers struggled making a true time commitment with the development teams to help not only clarify requirements but also help the team during the different phases of the SDLC, including testing. This lack of engagement forced the teams to make decisions on their own, resulting in the development of unnecessary or badly design features, a lot of project rework and significant delays in the project deadlines.

Interviewees also identified as one of their major challenges constraints in the availability of qualified resources. This goes from the number of people available to develop software in their teams compared to the overwhelming number of project requests they receive, to the talent of the resources themselves and the constant turnover that affected their ability to deliver projects successfully. Many resources, including managers and directors, had to play multiple roles in the organization, affecting their overall effectiveness in executing their assigned duties. This constraint usually affected the teams' ability to adapt and react to the many other changes they have to face on a day to day basis. As one of the interviewees indicated: "The

only real bottleneck we have in the [] team are resources, we just need more of us. The demand for what we are doing is really high and there is just so many of us".

The interviewees considered poor planning as the third challenge they most commonly had to deal with. This included not only bad execution within the planning phase of the projects, but also lack of planning from the decision makers, who would constantly change their priorities and would come up with last minute requests, expecting them not to cause any major impact in the current project deadlines.

In fourth place, the author found that organizational issues also represented one most common challenges mentioned by the participants. In this case the organization represents the institution they worked for, Purdue University. Here, the interviewees proceeded to explain how the lack of consistent standards to follow, excessive bureaucracy, the university's internal structure, as well as its slowness adapting to change, represented major constraints in their ability to deliver successful projects. Some of the interviewees also mentioned specific problems within the IT organization itself, including their lack of a central authority to determine common practices, their disconnect from the real customers (including faculty and students), as well as the number of groups that are extremely territorial about their scope and that are not willing to collaborate with groups who have ideas on how things could be done better.

Collaboration represented the fifth most common challenge found. The participants indicated that collaborating with other groups always represented a limitation in terms of timing, resources available and communication. As mentioned by one of the participants: "We are very siloed [...] software in general is meant to pull things together and tie things together, but we are in such vertical alignments in various places that that becomes really difficult". They also indicated that response

times from other groups could significantly impact their own schedules, and how, when necessary, the sharing of resources between groups was extremely difficult, in many cases because of the organizational challenges presented previously.

Many other challenges were also mentioned at a lower scale by the interviewees, including problems with scope definition (where stakeholders wanted to be provided with a solution that would solve every single problem as well as one that serves many purposes, resulting in an always changing scope); several process deficiencies (including having to circumvent obsolete university processes to be more efficient and deliver better service); lack of a supporting structure (where the parties involved in the project decision making never planned in advance for the maintenance cost the systems would have); poor change management (where no analysis is done on how different users would react to system changes, as well as a lack of understanding on what the real impact is for any decision and change made); lack of proper communication within the university (which includes project teams to customers and vice versa, as well as between different IT teams); poor prioritization (resulting in ever changing priorities); unclear governance (causing customers to be extremely confused on what the real process behind the approval of projects is), vendor constraints (in both the ability to adapt vendor applications to the university needs, as well as to the vendor's ability to solve different issues found), fierce competition (for those teams developing products that are more than information systems) and the inability to influence other parties to meet their deadlines so project commitments can be met.

In the next chapter the author proceeds to combine the information obtained during the literature review and data analysis chapters to create a selection tool that would be used to make a recommendation on a SDM or combination of SDMs that software development teams working for higher education institutions could use

to smooth their SDLC and address the multiple challenges they face while developing software solutions.

CHAPTER 7. DISCUSSION AND IMPLICATIONS

This research project had set as one of its objectives to determine what would be the most effective way for software development teams working for higher education institutions to develop software, while minimizing the limitations they most commonly find, and also while leveraging some of the advantages provided by the existing software development methodologies they use. In order to accomplish this goal, it became necessary to compare the characteristics of the different SDMs reviewed in the literature review chapter with the results obtained during data analysis phase of this research project.

To facilitate such comparison, the author needed to create an instrument that would allow them to determine, for each specific result obtained during the data analysis phase, the SDM that would better fit the needs of each particular topic. The creation of this selection instrument is explained in the first section of this chapter.

With the instrument in place, the author then expected to determine which SDM or combination of SDMs would better fulfill the needs of the teams working in software projects for higher education institutions, and derive a final recommendation based on the results. The review and analysis of the application of the selection instrument, as well as the resulting SDM are presented in the second and last section of this chapter, followed by a final validation of the results using the activity theory system.

7.1 Creation of the Selection Instrument

As mentioned in the literature review chapter, many are the SDM that have been created over the years. Unfortunately, up to this date, there is still not consensus among the software development and software engineering communities on which SDMs better adapt to the development of software projects, as well as on the best way to group and classify them for better understanding and selection. This lack of consensus became a crucial point during the creation of the selection instrument, given that the author needed to decide up to which level of detail they wanted the instrument to base on. If the author decided to create the instrument based on Kennedy (1998)'s approach, the level of detail the instrument would have would be significantly high, raising the possibility of obtaining scattered results. On the other hand, if the author decided to create the instrument based on Boehm and Turner (2003)'s approach, the level of SDMs detail included in the instrument would be much lower, increasing the chances of having more consolidated results.

For that reason, the author decided to create an instrument that merged both approaches into a single selection instrument. This would allow them to easily compare and determine if one approach would bring more benefit to the study than the other one by easily comparing all the results obtained after using the selecting tool.

When this approach was decided, the author then proceeded to select the topics out of each section covered during the data collection that received the most mentions by the interviewees. These selected topics were then used as the criteria through which the methodologies in both approaches would be compared against. The resulting selection instrument can be found in figure 7.1

With the instrument created, the author then proceeded to compare each one of the different methodologies against the defined selection criteria and mark,

		Software Development Methodologies					
Section	Topic	Linear Sequential Model/Plan Driven	Prototyping Model	The Incremental Model	RAD Model	Spiral Model	Agile Driven
Demographics	Teams are small						
	Projects are short in length						
Advantages	Ability to react to change						
	Helps to narrow scope						
	Helps to keep customer focus by getting timely feedback						
	Helps to keep customer focus by showing progress						
Challenges	Stakeholders - Understanding of the true need behind the request						
	Stakeholders - No clear scope definition						
	Stakeholders - Scope creep						
	Stakeholders - Not available when needed						
	Resources - Lack of talent						
	Resources - Lack of manpower						
	Planning - Poor project analysis						
	Planning - Unpredictable scope changes						
	Planning - Unrealistic deadlines						
	Planning - Hard deadlines based on academic/fiscal year						
	Organizational - Change is slow						
	Organizational - Lack of standards and guidance						
	Organizational - Disconnect from customers						
	Collaboration - Between different groups is difficult						
	Collaboraton - Sharing of resources between groups is hard						
	Collaboration - Response times are too slow						

Figure 7.1. Selection instrument

for each topic, if the SDM in review was a good fit to address the given need. The result of this exercise can be found in figure 7.2

Once all SDMs in the selection instrument were reviewed, the author proceeded to count how many of each one of the topics the SDM was a good fit for. The analysis of these results are explained in the next section of this chapter.

7.2 Resulting Methodology

With the results obtained from applying the selection instrument created in the previous section, the author was able to determine that none of the SDMs reviewed in this research project would be able to fully address the needs found in the data gathered from the interviews. The results were clearly scattered and ranged widely between methodologies. These findings highlighted the necessity for the design of a custom methodology that would address the needs of the teams

		Software Development Methodologies					
Section	Topic	Linear Sequential Model/Plan Driven	Prototyping Model	The Incremental Model	RAD Model	Spiral Model	Agile Driven
Demographics	Teams are small						X
	Projects are short in length						X
Advantages	Ability to react to change		X		X		X
	Helps to narrow scope	X		X		X	
	Helps to keep customer focus by getting timely feedback		X			X	X
	Helps to keep customer focus by showing progress			X	X	X	X
Challenges	Stakeholders - Understanding of the true need behind the request			X	X	X	X
	Stakeholders - No clear scope definition	X	X				
	Stakeholders - Scope creep	X		X		X	
	Stakeholders - Not available when needed		X				X
	Resources - Lack of talent	X		X	X	X	
	Resources - Lack of manpower						X
	Planning - Poor project analysis	X	X				
	Planning - Unpredictable scope changes		X			X	X
	Planning - Unrealistic deadlines	X		X	X	X	
	Planning - Hard deadlines based on academic/fiscal year	X					
	Organizational - Change is slow	X					
	Organizational - Lack of standards and guidance	X	X				
	Organizational - Disconnect from customers			X			X
	Collaboration - Between different groups is difficult	X		X	X	X	
	Collaboraton - Sharing of resources between groups is hard				X		X
	Collaboration - Response times are too slow			X			X
		10	**7**	**9**	**7**	**9**	**12**

Figure 7.2. Comparison of SDMs with selection criteria

under review, and that would serve as a guideline for them to use while working in their software development projects.

With this exercise it also became clear that, out of all the methodologies under review, the ones proposed by Boehm and Turner (2003) were the ones that had the most matches with the selection criteria, and were also complements of each other, in the sense that where one methodology lacked in ability to fulfill a need, the other one was able to do it. This was not the case with the SDMs suggested by Kennedy (1998), where the comparison among methodologies left important gaps in multiple parts of the selection criteria.

Given that each one of the findings in the data analysis could be addressed in one way or the other by parts and pieces of at least one of the SDMs categories suggested by Boehm and Turner (2003), it would not necessary to come up with a new set of principles and guiding rules from scratch for the new methodology to be

created. With an appropriate combination of the most important aspects of each methodology that addressed a particular need, the author would be able to develop a new methodology for recommendation. The creation of custom methodologies, according to Boehm and Turner (2003), is becoming a common practice among organizations of all kind, especially those who find themselves in the need of addressing the perplexity that surfaces while trying to select the SDM that would better address their organizational needs.

7.2.1 Recommendation for the Documentation of Requirements

During the review of the selection instrument results, the author was able to identify that software development teams creating solutions for higher education needed aspects of plan-driven methodologies, as mentioned by Boehm and Turner (2003), to be able to narrow the scope of their projects and avoid scope creep. Plan-driven methodologies characterize by explicitly and fully documenting the project requirements upfront, and by clearly defining the processes to follow in case a change in such requirements is needed. Unfortunately, the level of formality needed by plan-driven methodologies would be extremely hard to follow by the teams reviewed in this research, given the majority of the teams are very small (10 people or less), and they lack resources that could be dedicated to writing such deep levels of documentation. For these particular team characteristics, agile-driven methodologies would adapt better to their needs. Based on this, given that some level of documentation is needed to avoid scope problems, but at the same time the teams lack resources to write such documentation to higher extends, it is recommended for the teams to follow the use of user stories.

A user story, as defined by Ramsin and Paige (2008) "'defines a feature of the system as seen from the customers point of view. User stories are written by the

customer [...] and are nothing but short descriptions (about three sentences) of a certain chunk of functionality needed to be delivered by the system"'. The detail level that goes into user stories is just enough to allow team members to provide a semi reliable estimation of the time and effort needed for the implementation of the story. For such reason, the descriptions that go into the user stories remain very lean and to a very high-level of the requirements; yet they can be used to drive most of the planning, design and development activities of a project.

User stories, as mentioned before, would help the teams provide an early rough estimation of the amount of effort it would take them to complete the project, all of it backed up by the requirements provided in the story. This, from very early stages of the project, would also help the team deal with unrealistic deadlines imposed by the stakeholders, as well as with poor analysis sometimes made by the team itself, where they wrongly estimate the effort it would require to complete the project and do not realize of many aspects that have to be kept under consideration to make the project succeed.

Given that the interviewees also mentioned the lack of talented available staff as one of their major challenges, relying on tacit interpersonal knowledge is simply not an option for their teams, and explicit documented knowledge becomes a must Boehm and Turner (2003). With the use of user stories, teams would be able to keep memories of knowledge properly documented. These memories could be used not only in the present for the implementation of the issues, but they could also be used as a reference in the future by new team members, as well as other collaborators, when trying to understand the specifics of a system already implemented.

7.2.2 Recommendations for the Organization of Work

Continuing with the analysis of the selection instrument results, the author was able to determine that agile-driven methodologies would allow teams to easily adapt to change (a characteristic highlighted by many interviewees as critical for their success), as well as to properly react to unforeseeable changes in the project's scope. This, along with the need of keeping stakeholders highly involved in the project to answer questions and provide feedback, gave the author and indication that once the team is working on a project that has entered the development phase, they should define a short, regular cadence of work. This work cadence would allow them at the beginning of each cycle to determine which tasks they would be able to work on next (based on project priorities, as well as on any unforeseen changes in the scope and the project environment). At the end of these cycles, the teams would be able to meet with the stakeholders, show progress, gather feedback, and adapt quickly to the feedback obtained by incorporating it in the next work cycle or iteration. These interactions would, among other things, allow the teams to foster stakeholder ownership and involvement.

Based on these findings, sprints would be a good match for the teams under review. A sprint, as defined in agile-driven methodologies, is typically a short iteration of work that delivers an established amount of progress, satisfying a subset of the requirements defined for the project (Ramsin & Paige, 2008). Their length usually ranges from two weeks to a month and it involves a sprint planning session, where all parties involved in the project get to decide together what would be included in the sprint; the sprint development, where the team does the actual implementation of the agreed tasks; and a sprint review, where again, all parties involved meet to review progress, retrospect and adapt to existing or upcoming changes.

With sprints, cross-team collaborations and the sharing of resources would also become more predictable and simpler, and response times would clearly improve. With the implementation of these short cycles of work, each team involved in some sort of collaboration with another team would be able to raise a particular need and get it accommodated within the upcoming cycles of the other teams work, avoiding having to wait long periods of time for a response and the provision of the work needed.

7.2.3 Recommendations of the Level of Formality

Interviewees indicated that, from an organizational level, the biggest challenges they usually experienced related mostly to slowness around change, as well as the lack of standards to be followed. Although plan driven methodologies would alleviate the need of formality in standards and processes, they tend to contribute to the slowness around change, given the significant amounts of time needed for documentation and other levels of formality. On the opposite end, even though agile driven methodologies are great proponents of change and adaptability, their lack of formality around documentation and processes can cause perplexity among teams developing software applications. For these reasons, the author found a need for the inclusion of certain levels of formality in the processes followed while developing software for higher education institutions. Those processes, although needed, had to be to be small enough not to impact the speed of changes that need to happen and addressed promptly.

To not incur in an excessive amount of work for both the stakeholders in need of the software solution, as well as for the team in charge of developing the software, the author recommended a well-defined process at the beginning of the

software development lifecycle, as well as when changes out of the scope of the projects are being requested, and finally at the end of the project.

Several interviewees indicated that, during project initialization, one of the steps they struggled the most with was trying to get a clear idea of what exactly the stakeholders wanted out of a project they have requested, as well as trying to determine where in their priorities this project would fall. For such reason, the author proposes a formal mechanism of project requests, where stakeholders would have to clearly state the purpose of their project along with other meaningful information, including the benefit such project would bring, its impact, a timeline for when it would be needed as well as any other information considered appropriate by each team. With this formalized process, stakeholders would be forced to put more thought into their requests, and assess the true benefits of implementing their proposal.

Once a stakeholder has submitted their project request, their proposal would have to be reviewed and prioritized by an approval organism. This organism could be the team itself, a steering committee defined by the organization the team serves, or another arrangement of interested parties that together could determine not only if the project is viable and reasonable, but also where exactly this project would fall when compared with all the other priorities the software development team already has. The cadence on how often these reviews would happen, as well as how often the priorities would be reviewed, is left to be decided by each organization, but should be often enough to review proposals within just a few weeks from submission, in order to promptly determine the viability of the request.

If a project is approved and already in the process of being implemented, the author proposed the formalization of the steps through which stakeholders can request one or more major changes to the scope of the project originally requested.

Interviewees indicated in multiple occasions that one of the reasons why their projects often got delayed was due to the inclusion of new requirements to the project, without having them being formally reviewed and sized in order to determine how much of an impact such change would have in the originally estimated delivery date. With a formalized change request process, stakeholders would have to verbalize the reason behind the change in scope and justify its importance . Such reasons would have to be then reviewed and approved again by the organism that approved the project in the first place, and the team would have to provide a revised estimation of effort, which would include the changes in question.

Finally, at the end of each cycle, the author proposes a formalized process for the closure of the project. During this meeting or series of meetings, the development team along with the stakeholders, end users and any other parties involved in the project would get together to do a post-mortem review of the overall project experience, and would document both the parts of the process and the project that worked well, as well as those areas where improvement would be needed for future projects. As part of this session, all parties would declare the project considered as finished and any new versions of the system, as well as any other work necessary, would have to be handled through the change request process previously described.

7.2.4 A Proposed Software Development Methodology For Higher Education

After the analysis of the selection instrument results and in an effort to bring to life all the three elements discussed in the sections above, the author created a software development methodology as a proposal to be used by teams developing

software for higher education institutions. Such methodology can be found in figure 7.3.

7.3 Methodology Validation Through Activity Theory

In order to validate how well the different aspects of the proposed methodology would help software development teams alleviate the most common challenges they face while developing solutions for higher education institutions, the author proceeded to apply activity theory analysis to the proposed methodology by mapping it with the activity system model. The results of this exercise can be found in figure 7.4

With the results of this analysis it became clear that the new methodology has a well defined activity system, making clear the roles of the subject (software development team), the community (project, IT and external stakeholders, end users and the team as well), the object (the project itself) and the desired outcome (a system that fulfills the stakeholder needs). It also became clear that the activity structure is well formed, allowing to easily identify the different actions and operations needed at each step of the activity, something that the existing processes used by the teams lacked or had significant gaps in.

With the analysis it was also found that the definition of mediators in terms of tools, rules and roles had been properly defined. Even though some of the existing processes and methodologies had some well-established mediators, they did not necessarily have one of each, while others had mediators defined at a more ad-hoc basis, often causing confusion in the different parties involved.

The system dynamics of the proposed methodology were also properly defined, leaving nothing to surprise. The relationships between members are explicitly set, the context in which each one of the steps of the methodology must

be followed is clearly stated, and the expected result of each interaction is explicit in the model. Some of the methodologies currently used by the teams included in the study, as explained by the interviewees, had not clearly set the dynamics of the activity, leaving multiple inconsistencies between projects executed by the same team.

Based on this results, the software development methodology seems to be well found and properly designed, based on the principles behind activity theory and its defined system.

7.4 Summary

In this chapter, the author proceeded to analyze the results obtained from the data collection and analysis phases of this research project. After this and with the creation of a selection instrument that would help the author determine the SDM or combination of SDMs that would better adapt to the needs of software development teams in higher education, the author then proceeded to describe what they considered the most appropriate factors to be kept in mind for a new SDM. This new SDM was mean to help to alleviate the most common challenges faced by teams in higher education, while at the same time leverage the advantages of the existing methodologies in use. With all this data, the author proceeded to make a recommendation for a SDM that could be used by teams in higher education and presented such methodology in the last section of the chapter. Such proposal was then reviewed with the help of activity theory in order to validate its form and foundation.

in the next chapter, the author proceeds to summarize the findings of this research work and make some final recommendations.

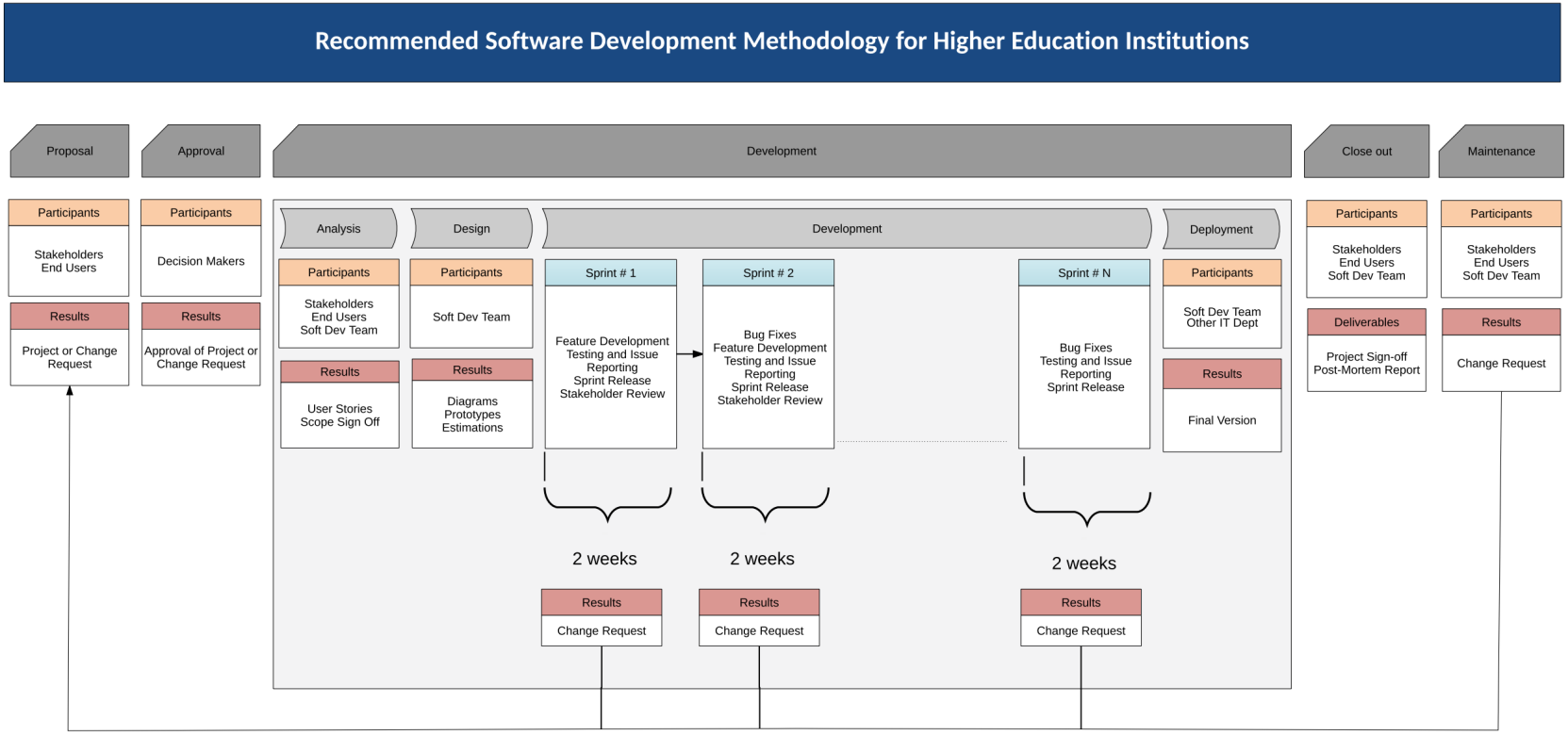

Figure 7.3. Recommended software development methodology for higher education institutions.

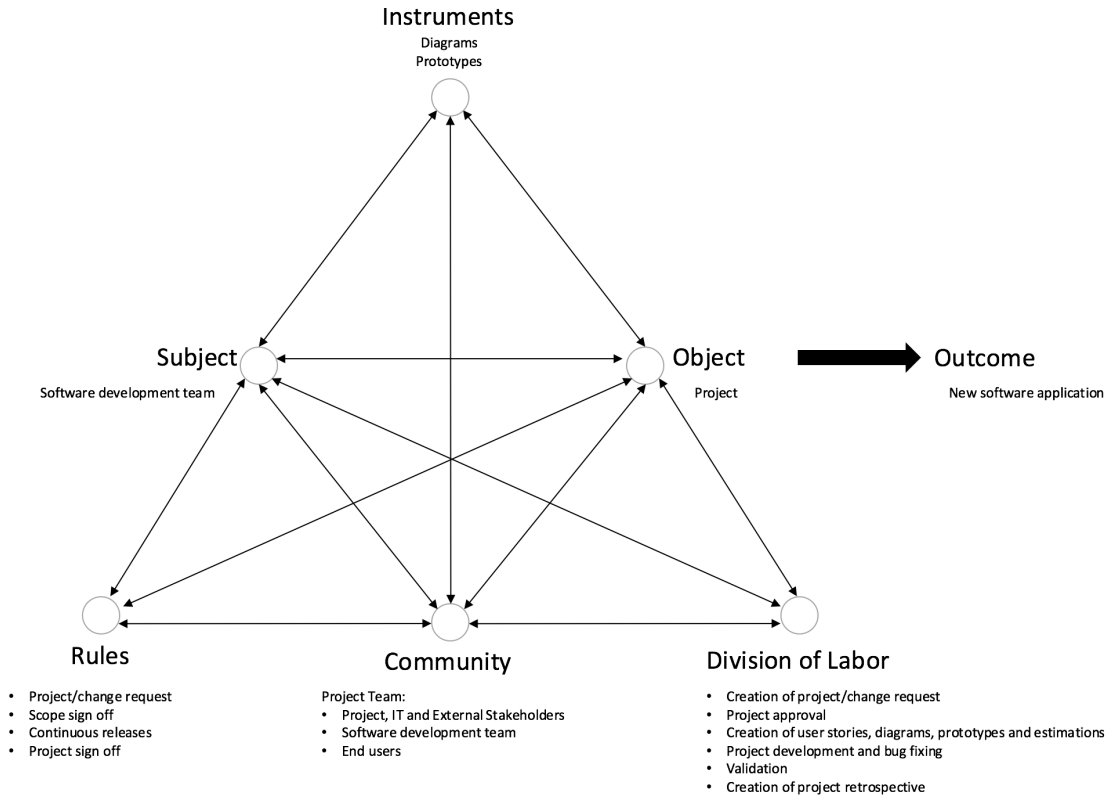

Figure 7.4. Activity system model applied to proposed methodology

CHAPTER 8. CONCLUSIONS

As stated at the beginning of this study, the purpose of this research was to analyze the opportunities and challenges the different SDMs used in higher education had, and to recommend a SDM that could be adopted by software development teams working for those institutions. Such methodology should be able to help them overcome the challenges they experience most commonly during their day-to-day operations, as wells as to adapt well to their particular team characteristics.

Throughout this research project, the author explored multiple existing SDMs, and assessed their different advantages and challenges. They also tried to identify any literature work that demonstrated the existence of a methodology that had been tested in higher education and that had demonstrated to meet the needs in all areas of such institutions, but up to this date, no such work had been done. Because of that, the author developed a new SDM, based on data collected from interviewing software developers and manager/team lead/directors from Purdue University, a land grant institution located in West Lafayette, Indiana. This SDM was developed step by step following the findings of the interviews, addressing the different needs mentioned by the interviewees during the data collection phase by utilizing parts and pieces of existing SDMs that were identified to be successful in those particular areas. The resulting SDM was then validated with the help activity theory analysis.

The world of software development, as we have known it traditionally, has been drastically changing (Boehm & Turner, 2003). Software systems are being

imagined and created differently than just a few years back. And as part of this evolution, software development teams are constantly trying to evolve the way they meet their goals while trying to keep up with this trend. The selection of the instruments that would help them succeed is not an easy task, given the infinite number of available options. For software development teams working for higher education in particular, this perplexity increments by the myriad of other different challenges they face while developing software for their particular field.

As it was analyzed in this study, these software development teams have tried to establish a way of developing software that works best for them, and their current processes vary widely from one to another. Unfortunately, up to this date and regardless of their current efforts to follow particular industry practices, these different approaches have not been able to address all their needs. Based on the findings of this study, the author believes one of the reasons behind their lack of success relies on the fact that these teams have tried to follow methodologies as presented in their theoretical description, and have not tried to combine methodologies to meet their particular challenges. In their book, Boehm and Turner (2003) make a case for the importance of finding middle ground between methodologies to better match an organization's needs, particularly between those organizations that follow plan-driven or agile-driven methodologies. Boehm and Turner (2003) incites organizations to use their common sense when evaluating the value behind each approach, and highlights the importance of combining discipline driven methods with agile driven ones, in order to adapt to the fast pace of technology.

Learning from Boehm and Turner (2003) insights and by reviewing the results of the data analysis done for this study, it became clear to the author that a mix of methodologies would be the best way for software development teams in

higher education to proceed with their projects, and based the methodology proposed in this study in all this information. With the proposed approach, the teams in review will have a tool at hand that has been catered to their particular needs and that they can take on an adapt as the software industry keeps evolving.

It is important to note that the adoption of the proposed methodology would probably have to start as a grass roots initiative by the software development teams. As some of the challenges mentioned during the interviews held for this study, a lack of consistency in standards, as well as the missing presence of a central authority at the university who can dictate some of these rules, forces teams to have to make this type of decisions on their own. This is contradictory to what many other types of organizations do, where the way of doing work by the software development teams is clearly dictated from an executive level. Even though the lack of guidance can be explained as a way of empowering the different teams, the results of this study highlighted the need seen by the teams to have a clearer vision on how software should be developed within the institution.

Finally, it is important to mention that, for a tool or methodology like the one proposed in this research study to succeed, it will become vital for the teams to properly educate their different stakeholders into the different steps they are expected to participate, and set the expectations clearly from the beginning of each project. Without stakeholder buy in and participation, the methodology here proposed would not succeed.

8.1 Future Research Recommendations

The SDM created and presented in the results section of the previous chapter is yet to be tested and evaluated by software development teams at Purdue University in order to determine its value, accuracy and validity of results. It is

recommended that, as the methodology is put to practice by the different teams, further revisions and iterations of the methodology are created until a refined version reaches a level of maturity high enough that significantly increases the chances of success for in-house projects at the institution. It is also recommended that, upon verification of success of the presented methodology at Purdue University, the SDM presented in this research project is also taken and applied to other universities to validate its generality.

In addition, future work could be done utilizing the same results obtained in this study, but modifying the selection instrument in such a way that each one of the selection criteria has a weight to determine its level of importance. This would differ from the work done by the author in this research where each selection criteria were weighted the same, without putting extra emphasis in the criteria that seemed to be of more importance to the participants. The results of this exercise could then be compared with the ones presented in this study, and an analysis of the discrepancies could provide some insight in future improvements to be made to the proposed SDM.

Based on the information gathered, the biggest majority of the teams included in this research work had ten or less team members. The author recommends further study where a body of teams whose size is bigger than ten is selected and have them apply the proposed methodology to analyze how well it would adapt to different team sizes as well.

Finally, once a sizable body of results is obtained from applying the recommended methodology to multiple projects in higher education, a quantitative analysis could be done as a further expansion of this work. Such work could help compare the project success rates of teams using the suggested methodology versus

those who are using other existing methodologies to corroborate the benefit behind utilizing the proposed methodology.

APPENDICES

APPENDIX A. INVITATION TO PARTICIPATE IN RESEARCH PROJECT

Dear [Participant Name],

First of all, I hope this email finds you well. As part of my thesis work, I am currently doing a research study that will analyze the software development methodologies and the challenges software development teams face while developing systems for higher education. This data will help me understand what different software development groups on campus are currently doing in order to successfully complete their projects and then try to identify a better way for teams in higher education to increase their project success rate while minimizing the number of challenges they face. As part of this analysis, I would like to speak with several software development managers and developers at Purdue, to get their perspectives on different topics.

Given your current involvement in the development of software at Purdue, I was wondering if you would be interested in participating in a 60-minute, voice-recorded, phone interview with me on a date and time to be agreed upon. During this interview I will be asking you to share information about the work that you do at Purdue, as well as the different processes and practices followed by you in your day to day activities, in order to complete any software initiatives assigned. This discussion will also include a few questions about the advantages and disadvantages of following such processes, as well as working with the different stakeholders most commonly involved in each initiative.

All information shared during the interview will **not** be personally identifiable and your participation in this study will be completely anonymous and voluntary. You may choose not to participate and, if you agree to participate, you can choose not to answer any particular questions asked, as well as to withdraw your participation on this research at any time during the process.

If you are interested in participating, please indicate which dates and times would fit your schedule better and then we can proceed to coordinate the logistics of the interview.

In advance, thank you very much for your time and help.

Best regards,

Daniela Rivera Alvarado
Graduate Student
Computer and Information Technology
College of Technology
riverad@purdue.edu
Phone: (765) 337.2148

APPENDIX B. INSTITUTIONAL REVIEW BOARD APPROVAL

PURDUE UNIVERSITY

HUMAN RESEARCH PROTECTION PROGRAM
INSTITUTIONAL REVIEW BOARDS

To:	ALEJANDRA MAGANA DELEON KNOY
From:	JEANNIE DICLEMENTI, Chair Social Science IRB
Date:	08/19/2016
Committee Action:	**Determined Exempt, Category (2)**
IRB Action Date:	08/19/2016
IRB Protocol #:	1608017993
Study Title:	Towards a software development methodology for projects in higher education institutions

The Institutional Review Board (IRB) has reviewed the above-referenced study application and has determined that it meets the criteria for exemption under 45 CFR 46.101(b).

Before making changes to the study procedures, please submit an Amendment to ensure that the regulatory status of the study has not changed. Changes in key research personnel should also be submitted to the IRB.

Please retain a copy of this letter for your regulatory records. We appreciate your commitment towards ensuring the ethical conduct of human subject research and wish you well with this study.

Ernest C. Young Hall, 10th Floor • 155 S. Grant St. • West Lafayette, IN 47907-2114 • (765) 494-5942 • Fax: (765) 494-9911

LIST OF REFERENCES

LIST OF REFERENCES

Bardram, J. E. (1997). *Plans as situated action: An activity theory approach to workflow systems*. Datalogisk Afdeling.

Barthelmess, P., & Anderson, K. M. (2002). A view of software development environments based on activity theory. *Computer Supported Cooperative Work (CSCW)*, *11*(1-2), 13–37.

Berg, B. L. (2009). *Qualitative research methods for the social sciences* (7th ed.). Boston, MA: Pearson.

Boehm, & Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Boyatzis, R. E. (1998). *Transforming qualitative information: Thematic analysis and code development*. Thousand Oaks, CA: Sage Publications.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77–101.

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

Döweling, S., Schmidt, B., & Göb, A. (2012). A model for the design of interactive systems based on activity theory. *Proceedings of the 2012 ACM Conference on Computer Supported Cooperative Work*, 539–548.

Dyck, S., & Majchrzak, T. A. (2012). Identifying common characteristics in fundamental, integrated, and agile software development methodologies. *Proceedings of the 45th Hawaii International Conference on System Sciences*, 5299-5308.

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of Management Review*, *14*(4), 532–550.

Eisner, E. W. (1991). *Taking a second look : educational connoisseurship revisited*. National Society for the Study of Education.

Engestrom, Y. (2000). Activity theory as a framework for analyzing and redesigning work. *Ergonomics*, *43*(7), 960-74.

Engeström, Y. (2015). *Learning by expanding: an activity-theoretical approach to developmental research* (Second ed.). Cambridge University Press.

Farrugia, A., & Al-Jumeily, D. (2012). The design, implementation and evaluation of a web-based student teachers' eportfolio (step). *2012 International Conference on Education and e-Learning Innovations (ICEELI)*, 1-7. doi: 10.1109/ICEELI.2012.6360612

Fjeld, M., Lauche, K., Bichsel, M., Voorhorst, F., Krueger, H., & Rauterberg, M. (2002). Physical and virtual tools: Activity theory applied to the design of groupware. *Computer Supported Cooperative Work (CSCW)*, *11*(1-2), 153–180.

Glass, R. L. (2004). Matching methodology to problem domain. *Communications of the ACM*, *47*(5), 19–21.

Hannan, M. (2011). Analysis of the collaborative activities in software development processes from the perspective of chronotopes. *Computers in Human Behavior*, *27*(1), 248–267.

Holcombe, W. M. L. (2008). *Running an agile software development project.* Hoboken, N.J.: Wiley.

Ibrahim, R., & Oxley, A. (2010). Proposed development methodology for higher education and library mash-ups. *2010 International Symposium in Information Technology (ITSim)*, *1*, 1–6.

Kamat, V., & Sardessai, S. (2012). Agile practices in higher education: A case study. *Proceedings of the Agile India 2012 Conference*, 48–55.

Kaptelinin, V., & Nardi, B. A. (2006). *Acting with technology.* Cambridge, MA: MIT Press.

Kaptelinin, V., & Nardi, B. A. (2012). *Activity Theory in HCI: Fundamentals and reflections.* San Rafael, CA: Morgan & Claypool Publishers.

Kennedy, D. M. (1998). Software development teams in higher education: An educator's view. *FlexibilITy: The next wave*, 373–385.

Kuutti, K. (1996). Activity theory as a potential framework for human-computer interaction research. *Context and consciousness: Activity theory and human-computer interaction*, 17–44.

Levy, Y. (2008). An empirical development of critical value factors (CVF) of online learning activities: An application of activity theory and cognitive value theory. *Computers & Education*, *51*(4), 1664–1675.

Lynch, J. (2013). *Chaos manifesto.* The Standish Group. Boston.

Magdaleno, A. M., Werner, C. M. L., & Mendes de Araujo, R. (2012). Reconciling software development models: A quasi-systematic review. *Journal of Systems and Software*, *85*(2), 351–369.

Mahanti, R., Neogi, M., & Bhattacherjee, V. (2012). Factors affecting the choice of software life cycle models in the software industry-an empirical study. *Journal of Computer Science*, *8*(8), 1253.

Martin, P. Y., & Turner, B. A. (1986). Grounded theory and organizational research. *The Journal of Applied Behavioral Science*, *22*(2), 141–157.

Matijasevic, B., Roncevic, H., & Orel, O. (2007). Agile software development supporting higher education reform. *29th International Conference on Information Technology Interfaces*, 375–380.

Nardi, B. A. (1996). *Context and consciousness: Activity theory and human-computer interaction*. MIT Press.

Nardi, B. A. (1997). Studying context : a comparison of activity theory, situated action models, and distributed cognition. MIT Press.

O'Leary, D. E. (2010). Enterprise ontologies: Review and an activity theory approach. *International Journal of Accounting Information Systems*, *11*(4), 336-352.

Patton, M. Q. (2002). *Qualitative evaluation and research methods*. Thousand Oaks, CA: Sage Publications.

Pavolka, R., Mount, V., Neymeyr, A., & Rhodes, C. (2005). From waterfall to rapid prototyping: supporting enterprise-wide adoption of the oncourse collaboration and learning (CL) environment at Indiana University. *Proceedings of the 33rd Annual ACM SIGUCCS Fall Conference*, 312–319.

Ramsin, R., & Paige, R. F. (2008). Process-centered review of object oriented software development methodologies. *ACM Computing Surveys (CSUR)*, *40*(1), 3.

Redmiles, D. (2002). Introduction to the special issue on activity theory and the practice of design. *Computer Supported Cooperative Work*, *11*(1/2), 1-11.

Sam, C. (2012). Activity theory and qualitative research in digital domains. *Theory Into Practice*, *51*(2), 83–90.

Tan, S., & Melles, G. (2010). An activity theory focused case study of graphic designers' tool-mediated activities during the conceptual design phase. *Design Studies*, *31*(5), 461-478.

Trevvett, D. (2013, August). *Enterprise application projects in higher education* (Research report). EDUCAUSE Center for Analysis and Research.

Vavpotic, D., & Bajec, M. (2009). An approach for concurrent evaluation of technical and social aspects of software development methodologies. *Information and Software Technology*, *51*(2), 528–545.

Woodside, A. G. (2010). *Case study research: Theory, methods, practice*. Emerald Group Publishing.

Yin, R. K. (2009). *Case study research: Design and methods* (Vol. 5). Sage.