Selecting a Software Development Methodology based on Organizational Characteristics

BY

Adrienne Farrell

An essay submitted in partial fulfillment Of the requirements for the degree of

MASTER OF SCIENCE in INFORMATION SYSTEMS

Athabasca, Alberta

October, 2007

© Adrienne Farrell, 2007
Dedication

This paper is dedicated to my husband, Chris Mintz. Without his support this project would not have been completed.
Abstract

This essay provides a general guide for Organizations to select an effective Software Development methodology based on their organizational characteristics. A Software Development methodology is the model an organization uses to develop software and includes the process, tasks, and activities necessary for successfully developing software projects within specific project constraints such as time, cost, and resources. The activities required to produce software include project planning, development of requirements, development of specifications, technical design or architecture, software configuration, software development, testing, documentation, training and support, and maintenance. This essay provides an overview of Software Development methodologies and presents a high level analysis and evaluation of each methodology. An overview is presented of each individual methodology where the methodology is summarized and its strengths and weaknesses are presented. Each development methodology is compared with the others and an overview is provided on the potential grouping of methodologies. Information is provided regarding the types of projects that are suited to the methodology as well as those that are not well suited. Several different software development methodologies are reviewed including waterfall, spiral, agile, XP, certification based methods such as CMM and ISO, and formal methods.

This essay also examines organizational characteristics and structures. Characteristics related to the bureaucratic nature of the organization are explained and analyzed as are organizational structure, product complexity, work effort, work type,
change philosophy and management, and organizational size. Given all of these characteristics some basic organizational types are presented including entrepreneurial, innovative, machine, diversified and professional. Each software development methodology is examined and information is presented regarding which methodologies are most and least useful within certain organizational structures. Details are presented regarding the applications, platforms, policies, and technologies that are best suited to each methodology. Software methodologies are examined within the context of organizational information system architecture. The guide would not provide one single solution which would apply to all projects and organizations but would allow organizations to base this selection on their individual organizational characteristics.
Acknowledgements

I would like to thank my husband, family, and friends for supporting me in this work.

I would also like to thank those at Athabasca who helped me produce this paper – particularly Richard Huntrods for supervising and Jackie Terrian for helping me navigate the administrative waters of the program.
List of Tables

Table 1: Organizational Characteristics and Type................................................................. 51
Table 2: Methodologies and Organizational Structures ....................................................... 56
Table 2: Methodology Compatibility.................................................................................... 58
# Table of Contents

DEDICATION ............................................................................................................................................... I

ABSTRACT ................................................................................................................................................. II

ACKNOWLEDGEMENTS ....................................................................................................................... IV

LIST OF TABLES........................................................................................................................................V

TABLE OF CONTENTS........................................................................................................................... VI

1. CHAPTER ONE - INTRODUCTION .............................................................................................. 1
   1.1 RESEARCH TOPIC.............................................................................................................................. 1
   1.2 RESEARCH GOAL .............................................................................................................................. 2
   1.3 DEFINITION OF TERMS ...................................................................................................................... 3
   1.4 IMPACT AND SIGNIFICANCE .............................................................................................................. 4
   1.5 LIMITATIONS..................................................................................................................................... 6
   1.6 ORGANIZATION OF THIS PAPER ......................................................................................................... 6

2. CHAPTER TWO - SOFTWARE DEVELOPMENT METHODOLOGIES................................. 8
   2.1 SOFTWARE DEVELOPMENT METHODOLOGIES - DEFINITION .............................................................. 8
   2.2 SOFTWARE DEVELOPMENT METHODOLOGY USAGE OVERVIEW ..................................................... 10
   2.3 THE WATERFALL METHOD .............................................................................................................. 10
   2.4 THE SPIRAL METHOD ...................................................................................................................... 14
   2.5 FORMAL METHODS .......................................................................................................................... 18
   2.6 AGILE METHODS .............................................................................................................................. 21
   2.7 EXTREME PROGRAMMING METHOD ........................................................................................... 25
   2.8 PROTOTYPING METHODS .............................................................................................................. 27
   2.9 RAD METHOD ................................................................................................................................3 0
   2.10 CMM/CMMI ................................................................................................................................3 3
Chapter One - Introduction

1.1 Research Topic

This paper reviews software development methodologies and presents information to assist in analyzing the suitability of a given methodology to specific projects and organizations. This information is designed to assist organizations in making the decision regarding which software development methodology to use during their software development lifecycle. Making the correct decision regarding what software development methodology to use can determine if projects are delivered successfully. A successful project is defined as a project that is delivered within schedule, within cost, and meeting all the project requirements. There are many risks on a typical software project and software projects may fail due to late delivery, cost overruns, or failure to meet customer requirements. The lack of an organizational software development process can increase the risk that a specific project will fail. When an organization follows documented processes such as a defined process for software development there is a greater chance that this organization will be able to consistently deliver successful projects.

This essay examines several different software development processes and methodologies. First traditional methods such as waterfall and spiral are reviewed. Then prototyping is examined and subsequently iterative methods such as Agile and Extreme Programming are reviewed. In addition, formal methods such as cleanroom are described. Consideration is given to other general methodologies such as Rapid
Application Development. In addition to the examination of these software
development methodologies, an analysis will be made of popular certification based
processes such as ISO 15504 (SPICE), Six Sigma, and CMMI. This essay proceeds
with analyzing the concept of the organization and defining its characteristics. These
organizational characteristics are examined in greater detail and an overview of
different organizational types based on these characteristics is presented. Once the
software development methodologies and organizational types are described this essay
reviews the compatibility and incompatibility between specific development
methodologies and organizational structures.

1.2 Research Goal

The objective of this research is to present a general survey of software
development methodologies and examine them in relation to organizations. The main
purpose of this essay is to provide a guide to assist organizations in selecting the
appropriate software development methodology to ensure that they can deliver
projects successfully. This paper focuses on a detailed overview of each software
development methodology including an examination of the methodology, its
advantages, its disadvantages and a review of the types of projects that work best with
a specific methodology. The goal of the research is to help organizations identify what
software development methodology should be used for specific projects to facilitate
delivering projects within schedule, within cost and meeting all the project
requirements.
The key to the successful implementation of a software development methodology is to match the methodology to the project and organization. Adam Kolowa states the following in “Which Development Method is Right for your Project?: “In my opinion, you can gain a lot more from retiring the “Which development process (if any) is best?” question forever, and replacing it with a slightly different one: “Which development process is best for my project?” (Kolowa, 2007). This essay will include an overview of software development methodologies along with the benefits and draw backs of each method. The results can be used by organizations to determine which methodology would be best suited to that specific organization. The positive outcome of this essay is the production of a reference guide for software development organizations which may assist in deciding on which software development methodologies to use on a project. The guide would not provide one single solution which would apply to all projects and organizations but would provide organizations with the tools they require to select a development methodology and work to deliver successful projects.

1.3 Definition of Terms

A software development methodology can be defined as a set of practices or processes that an organization follows to produce software. The term software development methodology is described by Chapman as follows:

“The documented collection of policies, processes and procedures used by a development team or organization to practice software engineering”
It is important to note that the software development methodology does not simply encompass the development of software code; it covers the entire process involved in developing a software product. The software development process incorporates the process through requirements, design, implementation, operation, maintenance and retirement of software. This process or cycle is referred to as an organization’s Software Development Lifecycle (SDLC). The SDLC is a model that covers the process of creating software from the creation of requirements, documentation, code, testing, release, to the maintenance and support of the product. Organizations use software development methodologies in order to work through the software development lifecycle and produce software products. The software development methodology must be formalized, documented and accepted as an organization’s standard in order to constitute an official, formal organizational SDM.

1.4 Impact and Significance

Research into software development methodologies is relevant to organizations looking for processes and methodologies that will improve quality, increase efficiency and reduce costs. Many organizations have difficulty delivering quality software products on schedule and within the allocated budget. Defining and following a formal software development methodology can increase the chances of project success. Organizations can benefit from research that delineates the strengths and weaknesses of various methodologies as well as analyzing which methodologies would best fit specific organizational structures and project types. The ability to matching the appropriate software development methodology to specific organizational
characteristics could save an organization time and resources. This problem is significant because if software projects fail this can cause organizations to lose money and potentially may affect the viability of the business. It is not just companies that create software as a product that face issues related to software development. Many companies who have core competencies other than software development have in-house software development teams to develop custom applications to facilitate the production of other products and services. The problems facing software development affect many different types of organizations and the benefits of using a formal software development methodology can benefit any organization which develops software.

The impact of not being able to match the software development methodology to the type of organization means that affected organizations may continue to have difficulty releasing software products successfully. The inability to release software may translate into cost and schedule overruns, quality issues, and project failures. For a company whose core competency is software development the failure of these projects can lead directly to the failure of the company. In other cases the failure of these projects can affect a company’s financial resources and success. Project failures may also affect employee morale and shareholder confidence. There is also a potential impact if the organization is using a software development methodology but it is not a good fit for the type of organization. The use of an inappropriate methodology may also waste time and resources and may contribute to project failure.
1.5 Limitations

This essay does not address methodologies that are specific to one part of the software development process. For example, specific techniques for developing software code are not addressed. This essay also does not examine specific technologies such as programming languages, software, or hardware. In addition specific project management tools are also out of scope for this paper. This essay also does not comprehensively cover every possible software development methodology or organizational type. The instances of both are too numerous to review in an essay. Instead specific organizational structures and software development methodologies have been selected based on which are most commonly used.

1.6 Organization of this Paper

This paper is organized into the following chapters:

- **Chapter One – Introduction**
  This chapter provides an introduction to this paper, an overview of the problem and the goal of this research.

- **Chapter Two – Review of Software Development Methodologies**
  This chapter provides an overall review of Software Development Methodologies and presents an analysis of literature on the subject.

- **Chapter Three – Review of Organizational Structures**
  This chapter provides an overall review of Organizational Structures and presents an analysis of literature on the subject.
• **Chapter Four – Selecting a Software Development Methodology based on Organizational Structure**

This chapter presents an overview of the recommended process for selecting a software development methodology based on organizational structure.

• **Chapter Five – Implementing a Software Development Methodology**

This chapter examines techniques for applying a software development methodology at an organization.

• **Chapter Six – Conclusion**

This chapter presents the conclusion to the essay and describes some potential areas for future research.
2. Chapter Two - Software Development Methodologies

This chapter reviews the software development methodologies which have been selected for examination by presenting an overview of each methodology along with the strengths and weaknesses of each methodology. This chapter also presents an analysis of literature related to each of the software development methodologies examined. This literature review includes analysis of the methodologies and covers the current problems encountered in project development – including an overview of some case studies. Literature and sources regarding different software methodologies are reviewed along with literature which provides case studies of the use of specific methodologies. Information is included regarding the suitability of each software development methodology to specific project types.

2.1 Software development methodologies - Definition

Software development is the act of producing software using a specific process. The Software Engineering Body of Knowledge (SWEBOK) describes the high level areas required to develop software as follows: Software Requirements, Software Design, Software Construction, Software Testing, and Software Maintenance (Abran, 2004). SWEBOK also describes the importance of the following components and groups within the software development cycle: Software Configuration Management, Software Quality, Software Engineering Management, Software Engineering Tools and Methods, and the Software Engineering Process (Abran, 2004). The high level process described by SWEBOK includes the stages for developing software from the
project’s first inception to its release and maintenance. Within the software
development process are several important activities including system requirement
definition and analysis, project scoping, architectural and detailed design,
requirements specification, development and coding, integration, installation, testing
and acceptance, training and documentation, implementation and maintenance (Green,
1998). A software development methodology describes how an organization uses a
process to perform the specific tasks required to develop software. The software
development methodology should include all the affected groups of the organization
as well as all the tasks required in order to produce, release and maintain software
products.

Software development methodologies are related to what the IEEE Computer
Society defines as software engineering. IEEE defines software engineering as
follows: “the application of a systemic, disciplined, quantifiable approach to the
development, operation, and maintenance of software” (IEEE, 1990). The key
information in this description is that the process for developing software is both a
systemic and repeatable process. A systemic process must cover the entire system,
which in this case is the organization. The process must be defined and documented
clearly so that it is repeatable for any project that is active within the organization.
This systemic process must also be proven capable of producing quantifiably quality
software product within the organization’s resource constraints. In order to create a
process that is systemic and repeatable several things are necessary including the
documentation of the process and the training of those who will be using the process
as well as the implementation and policing of the process. The process must also be quantifiable or measurable so that the success of the process can be measured and communicated.

2.2 Software Development Methodology Usage Overview

This essay provides an overview of several different software development methodologies. The methodologies that have been selected are those that are in use by organizations that develop software. The methodologies examined include waterfall, spiral, formal, agile, XP, prototyping, RAD and certification focused methods such as CMM/CMMI, ISO, and Six Sigma. There is also a brief overview of ad-hoc methods where no formal methodology is used. This essay looks at the usage of each of these methodologies independently and then some consideration is given to using them in combination. Organizations use many different types of methodologies and can use different methodologies for different projects or use methodologies in combination.

2.3 The Waterfall method

The waterfall method is a software development model which is based on following a sequential series of steps with very specific activities performed and deliverables produced at each stage. The stages in the waterfall method, presented in sequential order, are requirements, design, implementation/coding, testing and maintenance. The traditional waterfall method requires that one stage is complete and finalized before the project progresses to the next stage. Because of the emphasis on the sequential nature of the process the waterfall method has also been called the
linear sequential model. The waterfall method follows the philosophy of “big design up front” where the design and analysis work is completed at the start of the process and is not modified throughout the project. The waterfall method is rigorous and describes explicitly the steps to be followed and the sequence in which these steps must be performed.

The waterfall method has been in use since the 1970’s and is still in use today. There are also modified waterfall methods which take the basic waterfall structure and modify certain of its aspects. Three examples of this include the incremental waterfall model, the sashimi model, and Royce’s modified model. The incremental waterfall method changes the sequential nature of the waterfall method and assumes that some of the phases can proceed independently. It also allows for change and permits requirements to be handled incrementally. The sashimi method is a waterfall method where there is some overlap in the phases and these phases are permitted to occur simultaneously. Royce also presents a modified waterfall method which contains a repeating cycle between requirements and testing which passes through program design. This allows for some changes and refinements to the initial requirements.

The strengths of the basic waterfall model are that it spends more time up front in design and therefore projects that use this methodology produce a very complete and well understood requirements specification. This allows for more accurate time and cost estimates as there is less ambiguity to the project. Because the waterfall model works to have each phase complete before moving on to the next phase it is simple to track the progress through the project as the steps are very well defined. It is also more
straightforward to manage the resources of the project as there are not multiple activities taking place simultaneously. The waterfall model places an emphasis on documentation and, therefore, the documentation for the project is usually comprehensive and complete. The waterfall method is a disciplined and stable approach and can provide the project team with a strong sense that the project is managed and under control. The waterfall method is very efficient and minimized the planning throughout the cycle as all planning is done at the beginning of the project.

The weaknesses of the waterfall model are that it does not work well for projects that are not able to be fully designed or are susceptible to change as there is no opportunity to refine or change requirements once the requirements phase is complete. There is an argument that it is not possible to fully understand the system requirements until the project is into development and test – and that at this time requirements are often added or removed. The pure waterfall model does not allow for simultaneous execution of any phases and therefore resources which could be working on the project are not permitted to begin work until the prior phase is complete. This can extend the schedule for the release of the product. The waterfall method also does not support change management, project management or risk management methodologies. Some additional issues with the waterfall method are described as follows in the NASA overview of the waterfall method: “1. Problems are not discovered until system testing, 2. Requirements must be fixed before the system is designed - requirements evolution makes the development method unstable. 3. Design and code work often turn up requirements inconsistencies, missing system
components, and unexpected development needs. 4. System performance cannot be tested until the system is almost coded; under capacity may be difficult to correct.” (NASA, 2004) The waterfall method can also cause issues with teamwork and communication due to the rigid compartmentalization of the work. This can lead to a knowledge gap between members of the team and team work can suffer due to the clear divisions of responsibility.

The waterfall method is best for projects where requirements will not change and are able to be defined and frozen at the initial stage of the project. It is argued that the waterfall model and “Big Design Up Front” in general are suited to software projects which are stable (especially those projects with unchanging requirements, such as with "shrink wrap" software) and where it is possible and likely that designers will be able to fully predict problem areas of the system and produce a correct design before implementation is started. The waterfall model also requires that implementers follow the complete requirements design accurately to ensure that the integration of the system proceeds smoothly. The waterfall model is best suited to very traditional software development projects and is not suitable to projects that are more research and development oriented. The waterfall method can work well when team members – including the project manager - are inexperienced or do not have strong technical skills as it provides a good structure for an inexperienced team. The waterfall method also works well in a structured organization where there are formal approvals and milestones which can be tied easily to the sequential steps in the process.
The waterfall method is not suited to projects where requirements are unstable or where there are any opportunities for customer input to modify the requirements. Requirements must be fully signed off by all stakeholders prior to development and must remain unchanged. This can be a problem with custom software or any project that relies on customer feedback. The waterfall method is also not ideal for any projects which require rapid development as these projects often have a longer duration due to the sequential nature of the work. Waterfall does not work well when the project has changing expectations, budgets, or uses technology that is prone to change. In general, if there are any unknowns in a project then the waterfall method is not likely to be successful. The waterfall method is not generally suited to web based information systems, real time systems, event driven systems, or any applications that use new or leading edge technology.

2.4 The Spiral Method

The Spiral Method is based on the concept that a project should use incremental and iterative development. Iterative development includes a series of design-code-test cycles that are followed in frequent iterations. These iterations are used to design and develop the software and are used to refine the project requirements incrementally until all project requirements are fully implemented. Each cycle is complete when a subset of the requirements is fully designed, developed, and tested. Subsequent cycles then introduce additional incremental changes which are designed, developed and tested. This proceeds until all project requirements are complete. The philosophy of the spiral model is that organizations can reduce the risk to the projects by breaking
large projects down and executing the project in smaller deliverables. This allows for changes during the process and focuses the team on a smaller and more manageable work unit. The spiral method focuses is on minimizing the project risk, supporting changes to the project, and supporting manageable development of larger projects by deconstructing a project into smaller segments and allowing for analysis of risk in the development cycle.

The spiral model was created in the 1980’s as a response to the problems with the waterfall method. There are also several versions of the modified spiral method. For example, Barry Boehm describes a modified spiral in his article “Anchoring the software process”. Boehm’s model is based on avoiding problems with the spiral method including disappointed stakeholder expectations, gold-plating (the addition of features which are not requirements), inflexible solutions, high-risk downstream capabilities due to problems deferred from the early parts of the cycle, and uncontrolled developments. Boehm’s modified spiral method focuses on a risk driven approach that allows for custom modifications to the spiral process. Boehm indicates that the focus should be on the critical milestones of life-cycle objectives (LCO), life-cycle architecture (LCA), and initial operational capability (IOC). The LCO milestone is met when all stakeholders agree on the objectives, The LCA milestone is met when the system and the architecture are defined. The IOC milestone is met when the system has the site and users prepared.
The strengths of the spiral method are its ability to accept change to requirements throughout the development cycle. It is a more flexible methodology than the waterfall method as it allows for the requirements to be modified and refined as the project progresses. The spiral method also allows for feedback from stakeholders to change the requirements. Each iteration of the development cycle can include changes to the project and the project approach can be changed to match project requirements as part of these modifications. The early cycles are more cost effective and can absorb high impact changes more efficiently than if those same changes occurred later in the project. When changes occur early in the project even though costs will increase as well the overall risk associated with the project decreases as there is less chance that changes will need to be made later in the cycle when it is more expensive. (Norman, 2007). The spiral method can easily support prototyping and this can also help to control cost and risk – particularly for projects where there is high customer interaction. There are benefits to project and resource planning as resources can remain constant in a spiral model. The spiral model is extremely flexible and can work with other models to customize the model based on the characteristics of the project. The spiral model is easily adapted and customized to support specific project needs.

One of the main weaknesses of the spiral method is that it can be very difficult to control the changes coming into the system. Because this methodology is open to change it requires strong project management to control the changes so that the change is managed and limited. There should be a careful assessment of the impact of each change on the project. The project manager should also be comfortable rejecting
changes that are requested. The blind acceptance of all changes can be an issue with the spiral model therefore it is important to have a strong change management process in place. It may be difficult to have senior management and customers endorse the spiral model as they are often less comfortable with requirements and objectives that are not finalized. Project stakeholders may potentially require complete requirements in order to approve the project. There can also be problems with the spiral model when issues and problems are not addressed in the early cycles and are pushed to later cycles. This can be mitigated by strong project management. The spiral model requires an experienced project manager who can control moving from cycle to cycle, set deadlines and milestones, and control the change coming into the project.

The spiral model is best used for software development projects where the scope of the project is not well defined at the start of the project. These are often projects which are research and development based or projects that rely heavily on innovation or new technology. The spiral model supports change and can work well with projects where there are requirements for stakeholder feedback which may change the requirements. The spiral model also works well with larger, modular projects as it supports incremental development. However, the spiral model can become quite complex and does require a strong and experienced project manager. The spiral model is best used for projects with high risk and new technology where requirements are not well defined up front or where customer feedback or internal stakeholder feedback will likely change the requirements. Spiral also works well when implementation is more important than functionality as the constant integration allows for a fully
functional system throughout the project. The spiral model is not suited to software development projects where risk analysis and risk avoidance are not priorities. Spiral is also not a good match for projects that are difficult to build in increments – this can be true of larger projects.

2.5 Formal Methods

Formal methods are software development methods that use techniques based on mathematical statements as well as methods based on object modeling to specify software project requirements, develop software, and validate the software product. The philosophical foundation of formal methods is that using mathematical techniques as part of software product development improves the quality and reliability of the software that is developed. Formal methods use mathematical and logical reasoning and formal mathematical proofs to describe software. Formal methods are not generally used for complete systems but are very often used to validate small, critical parts of larger systems. Many developers are hesitant to use formal methods as it is perceived that formal methods can be difficult to understand and complicated to implement.

One example of a formal method is the Cleanroom method. The Cleanroom method was created by Harlan Mills and is based on allocating more time at the start of each project in an effort to prevent defects by using statistical methods to validate software quality and using formal methods to state the requirements. The following stages are the essential parts of the Cleanroom process: the process starts with
requirements analysis, followed by high-level design which defines the requirements as state machines, then detailed design, then coding by increment, then pretesting by increment, and finally statistical testing by increment. The Cleanroom method is also based on a philosophy of working to keep software defects out of the product. The emphasis in the Cleanroom method is to avoid having defects in the software and, if there are defects, to find these defects as early as possible and remove them when they are found. The belief is that it is less costly to find and remove defects early in the development cycle. Cleanroom uses formal notation to produce specifications which are the basis for both the software design and testing. As part of the software validation process review and analysis techniques are used on the software before it is run. The testing process is based on statistical analysis to focus on the errors that will have the highest impact on the system. Cleanroom bases the analysis of the software’s reliability on the mean time to failure (MTTF) and mean time between failures (MTBF). These two indicators are used as indications of the reliability of the software.

Proponents of the Cleanroom system and other formal methods believe that programs created using these techniques are more reliable and have fewer defects then programs which use other software development methodologies. The advocates of the Cleanroom method emphasize quality and they do admit that this methodology generally has no effect on project schedules. The Cleanroom method does not allow projects to complete earlier but does allow projects to have a higher quality. This is indicated by statements such as the following from the Software Engineering Institute: “Benefits of Cleanroom include significant improvements in correctness, reliability,
and understandability. These benefits usually translate into a reduction in field-experienced product failures, reduced cycle time, ease of maintenance, and longer product life” (SEI, 2007). Cleanroom is best used for software development projects in organizations that can support systematic design methods, formal code and design inspection, documented requirements, unit testing, configuration management and functional testing. Cleanroom does not preclude using other development and management methodologies and works well with methods that are compatible with the cleanroom practices.

Formal methods can facilitate defining project requirements and can also assist in preventing errors from occurring with specifications. Formal methods can also assist designers and stakeholders who do not have an expert understanding of the system by breaking the system down into formally specified functions. Several tools have been developed by those advocating formal methods to assist in the use of this technology in an endeavor to make these methods easier to use. Advocates of formal methods state that they provide more precise software specifications, higher quality internal communication, support for design verification before the test period, and higher quality and productivity. (Dorfman, 1996). Several proponents of formal methods indicate that an organization can use formal methods for part of a project even if it is not deemed appropriate to use these methods for the entire project. The general recommendation is to examine the requirements of each organization and fit formal methods into the software development methodology that will work best for that organization.
Formal methods are described as having the following weaknesses. They are criticized for not being useful in system validation. Formal methods can be used to verify a system but not to validate it. This indicates the weakness of formal methods in confirming the use of the system in the real environment. This also includes the issues related to testing ambiguous concepts such as the usability of the system. Another problem critics have identified with formal methods is that they rely on the machine as an abstract concept and not as a physical concept. This means that there may be issues with physical restrictions such as memory and resources which are not apparent in an abstract machine. Another critique is that formal methods can be complicated to introduce into an organization and there is a great deal of analysis work that must be done in order to strategize how to adopt the methods. It can be expensive both in terms of time and resources expended to implement formal methods as there is a great deal of education and training required in order to use formal methods effectively.

2.6 Agile Methods

The Agile software development method was developed as a reaction against methods such as the waterfall method which were perceived as “heavy” methods which required too much documentation and formal process and planning. The Agile method is fundamentally an iterative development methodology, based loosely on the spiral method but anchored to the Agile philosophy. This philosophy is articulated in the Agile manifesto (Beck, 2001) which specifies that the following are the most important concepts for a software project:
• Value individuals and interactions over processes and tools
• Value working software over comprehensive documentation
• Value customer collaboration over contract negotiation
• Value responding to change over following a plan.

The Agile manifesto advocates the importance of individuals, including focusing on strong working relationships among the team members and the need for team and individual motivation. Customer satisfaction is extremely important and the agile method stresses the need for close relationships between the project members and the customer. The software process itself must be flexible and include the ability to accept change as necessary. However, one of the fundamental principles of Agile is that the process must also take into consideration implementing software that is continuously in a working state.

Agile methods focus on the minimization of risk by developing small segments of the project using iterative techniques. Each iteration should last up to four weeks at the most. The iteration itself must encapsulate all of the components of the project including planning, requirements, design, development, and testing. Each project iteration implements a specific part of the project requirements and when the iteration is complete the product is in a status where it could be released if necessary. Once each iteration is complete and a new iteration is to begin the status of the project should be analyzed and any corrective action necessary should be taken. This analysis is performed by the entire team and changes are made to the process or product if necessary. Agile techniques concentrate on the team members and emphasize the
importance of “informal” face-to-face communication. Agile project planning within these iterations is designed to be team-focused and collaborative. The project planning involves the entire team and all team members must be actively involved in the project scheduling and priority analysis.

Advocates of Agile methods describe several advantages to using Agile methods. Agile methods produce a fully functional product after each iteration and these products can be easily demonstrated and even released if necessary. Agile methods accept change and can support changes to the product even quite late in the development cycle. Therefore, they can handle projects where requirements are not well defined or prone to change. Proponents of Agile methods declare that a project that uses Agile methods is more likely to meet stakeholder requirements and therefore more likely to have a higher customer satisfaction. This is mainly due to the focus on customer communication throughout the cycle. Agile methods can minimize some risks faced by larger projects as they force the project team to break the project down into smaller, more manageable segments.

Agile has been criticized for not having structure and not producing the required documentation to complete and maintain a product. Due to the lack of documentation, Agile methods can be problematic for any organization that is required to produce documentation artifacts for certification or legal reasons. Agile methods are often criticized as being closer to “ad-hoc” methods where individuals can control the process based on their own individual desires. When Agile methods are used it can be
difficult to analyze where the project will be in the longer term future due to the support for changes throughout the cycle. This can make it difficult to do long term, strategic planning. Agile projects can be susceptible to “scope creep” where features are continuously added throughout the cycle and the final requirements of the project keep shifting. This could make it difficult to define when to release the project. Critics have also indicated that it can be difficult to find the correct employees to make Agile methods work properly. Agile methods are acknowledged as requiring experienced and highly motivated teams.

Agile methods are best suited to experienced, dedicated teams working on lower risk projects. Agile methods are a good fit for a process where requirements are changing rapidly but are not as good a fit for systems that require finalized requirements early in the project or final requirement signoff prior to coding. In order to succeed with Agile an organization must have strong individual contributors but those contributors must be willing to work collaboratively. Agile methods therefore do not work well in organizations with communication issues or organizations with a great deal of internal politics. Agile methods are particularly suited to smaller development projects where the requirements are not known or are rapidly changing. The culture of the organization must support the Agile philosophy. For example, the organization should value the importance of team work as well as the ability of all team members to contribute to the project. Agile methods need not be used on their own for a project as it can be very effective to use Agile methods in combination with other methods.
2.7 Extreme Programming Method

Extreme Programming follows some of the same philosophies as Agile development - advocating evolutionary design, customer satisfaction, and teamwork. XP stresses the importance of the customer perspective and advocates being on site with the customer if possible. The main philosophies of XP include planning by creating user stories, managing small releases with continuous integration and refactoring, using a metaphor for the project, having developers work using pair programming, and emphasis on collective ownership of the project. XP projects work through four phases: planning, designing, coding, and testing. Planning involves designing the user stories, creating the project schedule, planning the iteration for each cycle. The importance of daily meetings as the main method of communication is stressed. The designing phase includes designing for simplicity, working with the metaphor for the system, and constant refactoring of the design. The coding phase is based on adhering to set coding standards, coding the unit tests first, using pair programming, integrating early and often, endorsing collective ownership of code, and leaving optimization until the end. The testing phase is based on all code passing unit tests prior to release, the creation of new unit tests every time a bug is found, and the running of acceptance tests created from user stories. XP has associated values built into the methodology including focus on communication, simplicity, feedback, courage, and respect.

The strengths of XP are that it is a lightweight process which assumes and accepts that change will occur in the system. XP is flexible and can easily adapt to
changes in the requirements. Watts Humphrey outlines several of the advantages of XP as follows: XP emphasizes the customer and therefore increases customer satisfaction. The fact that programmers must commit to the schedules leads to more accurate scheduling. The emphasis on quality and on responsibility means that the entire team works to produce a quality product. The emphasis on frequent and extensive testing also leads to a higher quality product. The constant integration and refactoring of the code means that the product is always in a functional state. A functional product can be demonstrated to the customer throughout the project. The fact that the product is always in a functional state minimizes the potential for project failure. Programmers state that XP allows them to focus on coding rather than spending time on administrative and project overhead such as paperwork and meetings.

XP is also criticized for several of its characteristics. Watts Humphrey comments on its main drawbacks as being that it does not work well for larger projects and the methodology does not specify what types of projects XP would work best with. Humphrey also mentions that XP has a weak design stage and that it can be difficult to obtain management support to move to XP because the transition can be difficult. XP is criticized on the quality front for not having a structured review process to reduce the defects and it seeking to add quality through testing rather than through design and review. Other critics have indicated that XP is a step back in process to a “code and fix” ad-hoc methodology and that organizations will move to XP when they want to move away from using any formal process. One of the main criticisms of XP is that it
is a difficult and challenging methodology to implement and that it takes a great deal of discipline to be able to follow the methodology properly. Criticism of the pair programming concept focuses on issues of lowered productivity and the personnel problems that can occur when people are forced to work in pairs.

The XP methodology works well in turbulent environments where there is a fast pace, a high level of change, and requirements uncertainty. It can also work well with smaller teams and projects. XP can also work well with new technology where the project may have many uncertainties. XP does not work well with larger projects as it is not possible to maintain the communication requirements of the methodology with a larger group. XP does not work well with projects that require contractual sign-offs, adherence to standards, or safety critical applications – in general due to the documentation required by these types of projects. XP is also problematic for very complex projects with many interdependencies. XP does not work well with co-located teams due to the requirements for pair programming and communication.

2.8 Prototyping Methods

The prototyping method is an iterative method based on developing a working model of the system called a prototype. Prototyping is generally not used as a standalone methodology but as part of a larger iterative methodology such as Agile or RAD. The prototyping method breaks a project into small segments and creates small prototypes of the product during each of these segments. The prototypes are
examined, analyzed, and refined until they meet customer requirements. The early prototypes in this cycle are most often developed and discarded. This method is based on the belief that requirements are not complete at the start of the project and can be added and changed throughout the project lifecycle. As these requirements change new prototypes of the system are developed. The prototyping method can include several different types of prototypes including GUI prototypes, limited functional prototypes, and existing system modification prototypes. The prototype is designed, created, and then assessed and refined up until it is complete. There are several different types of prototyping including model based prototyping, rapid prototyping, evolutionary prototyping, dynamic systems development method (DSDM), operational prototyping, and evolutionary rapid development (ERD).

One advantage of the prototyping method is that it provides a view into product functionality as well as product usability early in the cycle as well as throughout the cycle as changes occur. This view supports a process to change and refine the project requirements if necessary. Using the prototyping method allows for mitigation of project risk as the stakeholders have a chance to review the product to ensure it meets requirements. Another advantage is that the prototype provides a working replica which can be used for demonstration purposes. The prototype helps the users to make their needs clear by having them respond to a prototype. The prototype method requires all stakeholders to be involved throughout the cycle and therefore all stakeholders must understand and accept the product. This can improve the communication with project stakeholders. The prototyping method can also help
stakeholders to clarify requirements that are not clear by presenting several alternate prototypes and allowing the user to decide. The prototyping method encourages creativity and flexibility in product design. This method allows for a very quick development of a functional application.

One of the weaknesses of the prototyping method is that it is easy to not document the project properly. The prototyping system may suffer from lack of process control including documentation and lack of up front analysis and investigation as the goal is to get a prototype working as soon as possible. Another potential weakness is that prototype systems can often have performance issues as the development often does not focus on any non-functional requirements. Prototyping can be expensive if there are many development iterations. Continuous stakeholder feedback can lead to problems with change control and scope creep. Prototyping can lead to pressure to release early prototypes as products because the product appears to be complete – this is particularly a problem due to customer pressure. Early release of prototypes can lead to programs that are not fully functional, which have limited design, or are using inefficient development practices. Prototyping can provide false expectations, particularly related to project schedules, where the customer believes that the product is complete when it is in fact still a prototype.

The prototyping model is best used for software development projects that are new development projects rather than updates to existing systems. The prototyping method is useful for systems where there is extensive user interaction or where there
are many users, relationships, and functions. This model is also good with projects where the requirements are not known or are not clear. This method is also useful when requirements are constantly changing. Prototyping can help in projects where there is a great deal of pressure from stakeholders to have immediate, visible results. The prototyping method requires discipline at the management level, particularly an experienced project manager and experienced senior team members. The prototyping model is not suited to software development projects which are transactional or batch based – such as e-commerce systems. This method is also not useful in developing systems which are concerned with performance and scalability.

2.9 RAD Method

Rapid application development focuses on the quick development and delivery of a project at low cost and with high quality. RAD works to reduce overall project risk by deconstructing the project into smaller segments during which prototypes are created and refined. RAD supports change during the development process – change can be introduced at any time in the cycle. RAD is based on iterative prototyping, user participation, and the use of tools that facilitate rapid development. The foundation of RAD is to satisfy customer business requirements in a short amount of time. The project is tightly controlled by prioritization of features and the definition of “time boxes”. Time boxing is the process of moving features out to future releases when including the feature is compromising the schedule. The schedule controls the project and requirements are reduced to fit the time if necessary. The goal in a RAD project is to deliver the minimum set of requirements necessary in the shortest time period. The
customer is designed to be very actively involved in the system design to ensure customer satisfaction. RAD projects produce final software – not prototypes – but follow a prototyping method to do so.

The main advantage of RAD is that it shortens the time required to produce the product. The rapid product release ensures that there is less risk of the business changing during the product development cycle – such a change may mean that the product is no longer competitive. Another advantage is that the application will fully meet customer requirements and therefore customer satisfaction is high. There is a very early version of the product available as a prototype and the customer can see tangible project results very early in the cycle. RAD provides the users with a sense of ownership and commitment and customer satisfaction can be higher as the customers are involved throughout the cycle. The RAD method concentrates on the user and therefore the user has a high level of input into the system. RAD can support rapid changes to the system based on changing requirements or user demand. RAD can provide a large savings in time, cost, and resources.

The perceived weaknesses of the RAD system are that it does not deal well with packaged solutions and does not work well with teams that are not flexible. RAD requires customer participation throughout and customers can become frustrated with the constant iterations. The RAD method requires a strong and experienced project manager with a good technical understanding of the project to control the requirements on the project. (Mochal, 2001). There are several risks to implementing a
RAD methodology. It is possible that the system quality may be lower than with a traditional approach. Also, it is possible to miss information and requirements as the team is working quickly through project iterations. This may lead to requirement misunderstandings and miscommunications with the customer. It is possible that there may be additional requirements implemented that were not necessary as it can be difficult to reduce the requirements to the minimum necessary. There is risk related to inconsistencies within the system as requirements are often not considered systemically. RAD is low on reusability as it is very difficult to reuse modules created using a RAD method. Due to the quick, iterative cycles larger issues such as scalability and performance can be missed and may be problematic. One main problem, as with other iterative methods, is that there can be a tendency for problems to be deferred so that the project can show success early.

RAD works best with small or medium projects where the scope is well defined. RAD requires clearly defined user groups who have a very strong knowledge of the application. RAD can work well when the requirements are not well understood at the start of the project. The RAD project team should be stable and skilled and there is strong project management available. The architecture and the technology of the product should be clearly defined and all technical components must exist. The development team must be reasonably autonomous and able to make design decisions. RAD does not work well in large projects which involve distributed systems or with real-time systems. RAD does not work well with complex systems where there is a great deal of data. RAD requires firm business objectives and does not work well
when the business objectives are obscure. RAD does not work well with projects that require full functional requirements to be set before any code is written. RAD does not work well with large project teams or when there are many people required to make decisions.

2.10 CMM/CMMI

The Capability Maturity Model is a process model that was created by the Software Engineering Institute at Carnegie Mellon University to describe best practice processes for software development. The CMM has been retired and has been replaced with the CMMI. However, the CMM is the basis of the CMMI and it is worth examining the fundamentals of the CMM. The CMM outlines different levels of process maturity based on the characteristics of the software process. The five maturity levels are as described in the SEI book *The Capability Maturity Model: Guidelines for Improving the Software Process* as follows (SEI, 1999):

- Level 1 - Initial. The process is ad-hoc and chaotic with few processes defined. The success of a project depends upon individual heroics.
- Level 2 - Repeatable. The organization has some basic project management and is able to repeat the process to produce successful projects.
- Level 3 - Defined. The organization has a documented software process which is used throughout the organization for all projects.
- Level 4 - Managed. The organization measures the software process and project quality and both of these are controlled.
• Level 5 - Optimizing. Continuous process improvement supported by feedback.

The levels each have several process areas which delineate the focus of the organization at that level. These are referred to as the key process areas and include activities that the organization should be following if they are operating at that level. The CMM not only provides these levels to allow organizations to assess their level but also to support organizations improving their processes to move from one level to the next. The CMM is a good foundation for many organizations as it provides a useful tool for assessing where the organization is and a foundation for deciding how to improve software processes.

The CMM is a useful tool for assessing the process maturity of an organization. This can be very helpful for comparing different organizations and for determining where an organization is and what path the organization can follow to improve. CMM is generally accepted as a very useful framework for process assessment and improvement. It is well laid out and detailed and allows an organization to prioritize and plan process improvement. The CMM appears to work best for heavily bureaucratic organizations that can dedicate resources to auditing the organization. Organizations that are not as bureaucratic may feel that CMM is overly heavy and that this heavier process slows down the project development process. There is criticism that there is not much statistical evidence that CMM helps projects or organizations. Some users find fault with the fact that the CMM is mainly a self-assessment and that
organizations are not required to be CMM certified by a third party. This can lead to organizations falsely representing their CMM levels.

The Capability Maturity Model – Integrated (CMMI) is a reworking of the CMM based on lessons learned since the inception of the CMM. An important change is that CMMI has been built with cross-model support as one of its main goals. CMMI is designed to be used as an appraisal technique to examine the process maturity of an organization and provides a guide to best practices and a framework for organizing those practices. There are three different classes of appraisal (A, B and C) which have several different appraisal methods. The Standard CMMI Appraisal Method for Process Improvement (SCAMPI) meets all of the appraisal requirements for CMMI. The CMMI has two types: the staged version and the continuous version. The staged version uses levels and is similar to the implementation of the CMM. The continuous version uses capability levels rather than maturity levels. CMMI models have process areas (PAS) which contain goals comprised of practices. There are currently 22 process areas in CMMI version 1.2 (released in August 2006). There is a great deal of focus on how CMMI works with other processes such as Agile, Six Sigma, XP, etc.

Process improvement methods such as CMM and CMMI can help organizations to analyze issues with their existing process and determine the problem areas. The SEI states that “A process model is a structured collection of practices that describe the characteristics of effective processes. Practices included are those proven by experience to be effective.” (SEI, 2007). These process methods will not provide a
new development methodology but can help organizations to improve. This is particularly useful for organizations that have problems with missing deadlines, high development costs, low management visibility, quality problems, customer complaints, and poor morale. The basis of the CMMI is that the quality of a product and a system is based on the quality of the process used to develop the product (SEI, 2007). Process is presented as the piece of the development process that combines the technology and the people effectively. The CMMI can provide the means for an organization to start an improvement process, provide a common vision to an organization, and contribute a basis for prioritizing improvements and a technique for defining what improvement means. An organization could use CMMI to analyze the state of their processes and as a basis for the foundation needed to select and institute a software development methodology.

2.11 ISO/IEC 15504 - SPICE (Software Process Improvement and Capability Determination)

ISO 15504 is also referred to as SPICE (Software Process Improvement and Capability Determination) and is a method for assessing software processes. It is similar to CMMI in that it is not a methodology itself but provides a good foundation for determining the status of an organization’s methodologies. ISO 15504 has a reference model that contains both a process component and a capability component. The process component looks at several categories including customer, engineering, supporting, management and organization. The capability component is based on six capability levels as follows: Level 5 - Optimizing, level 4 - predictable, level 3 -
established, level 2 - managed, level 1 - performed, and level 0 - incomplete. Each process is measured using defined attributes where each attribute is assessed as not achieved, partially achieved, largely achieved, or fully achieved. ISO/EIC 15504 is used for assessing the level of an organization’s process and assisting in organizational process improvement.

ISO/EIC 15504 has several advantages over CMMI. The big difference between the two is that 15504 allows for individual processes to be examined and assessed. This can be beneficial for organizations that have the resources to make some change but cannot support enterprise wide changes. 15504 is easier for smaller companies to implement as it uses less company resource than CMMI. Rout describes the advantages of ISO 15504 over CMM as follows:

“The very fact that SPICE allows for the evaluation of individual processes such as project management and configuration management in an organization is a big advantage. SPICE’s treatment of acquisition is far better than CMMs… While CMMI is applicable to large organizations and involves an army of people for a longer time increasing the cost, SPICE is inexpensive and much less resource-intensive. It is a handy method to assess the capability of the software for small companies.” (Rout, 2001)

ISO 15504 has not generally been as successful as the CMM/CMMI for several reasons. The standard must be purchased from ISO while the CMMI is a free download. CMM was created prior to ISO 15504 and had gained acceptance from some groups prior to the ISO standard being developed. CMMI and ISO/EIC 15504 are compatible and can be used in combination. The SEI indicates that 15504 is not necessarily designed to be used as a standalone tool and is most effective when used in combination with other methods and models.
2.12 Six Sigma

Six Sigma was developed by Motorola to improve processes by working on the elimination of defects throughout the product development cycle. Six Sigma defines a defect as an area where a product does not conform to specifications. Six Sigma is based on the philosophy that the goal of a process is to reduce variation in outputs and that processes can be measured, improved and controlled. Six Sigma also emphasizes the importance of the entire organization working at process improvement – particularly senior management. Six Sigma is comprised of two methodologies: DMAIC (Define, Measure, Analyze, Improve, Control) and DMADV (Define, Measure, Analyze, Design, Verify). DMAIC is used to improve existing processes while DMADV is used for new processes. The basis of Six Sigma is that a systematic approach can be taken with a strong concentration of the voice of the customer. Six Sigma has professionalized the quality function by using martial arts terminology to define a hierarchy as follows: Yellow belt, Green belt, Black belt, Expert, Master Black Belt, Champions, and Executive. The term Six Sigma represents the standard deviation of a population – the goal being to have six standard deviations between the mean of a process and the nearest specification limit. If Six Sigma is implemented properly variation is under control and there are basically no items that do not meet the specifications.
Advocates of Six Sigma indicate that there can be large cost savings when Six Sigma is implemented. Motorola has described that they have “documented over $17 billion in savings to our own organization” due to the implementation of Six Sigma (Motorola, 2007). Another advantage is that Six Sigma reduces variability and therefore reduces defects. Criticisms of Six Sigma include that the Six Sigma metric is not a good measure of customer satisfaction, that it is quite complex and ambiguous, and that it does not differentiate between different types of processes. There is also substantial criticism that Six Sigma is used because it is fashionable. Also, Six Sigma is a “self certification” and companies can claim that they are Six Sigma compliant without any formal certification. Six Sigma is very structured and can prevent innovation and new project design from being successful. Six Sigma can be expensive to startup and require substantial resource allocation in order to succeed. It is not recommended for projects that rely on innovation. Six Sigma is best for organizations that are based in manufacturing, transaction based, research based or environments that require a low or zero defect rate.

2.13 Ad-Hoc Methods

Ad-hoc methods are also described as “no-process” methods. Ad-hoc is a generic term for the process followed when there is no formal method in place. Less mature organizations often do not have formal software development methodologies, however, they are able to complete projects and produce software products. Within these types of organizations the development is usually guided by the direction of the
key individuals who perform the work. This can devolve into the “code-and-fix”
method when the basic process followed is to develop and fix issues until the time
comes to release the product.

The failure or success of an ad-hoc project depends to a great extent on the skills
of the key project individuals. This process can work if there are very strong
individual contributors or a very strong team. However, the problems with ad-hoc
methods are numerous. Ad-hoc methods are not predictable and project timelines and
resources cannot easily be estimated. This makes it very difficult to estimate or plan
releases, resources, or budget. The success of these methods is dependant on
individual heroics and is therefore quite variable and is vulnerable to the knowledge
and desires of a single individual. Ad-hoc methods do not provide any room for long
term organizational maturity or growth. Many organizations follow a type of ad-hoc
method – often because they are not able to successfully follow a specific
methodology for a project. Ad-hoc methods appeal to “cowboy” developers who
prefer to work with little to no guidance. At an organization where there are no strong,
skilled individuals to support the development this is an extremely ineffective and
immature method.
3. Chapter Three - Organizational Structure

This chapter provides an overall review of Organizational Structures and presents a general review of some of the literature on the subject.

3.1 Definition

An organization can be defined as “a group of people intentionally organized to accomplish an overall, common goal or set of goals” (McNamara, 2007). The important components of this definition are that organizations are groups of people and that this group of people is brought together to accomplish specific actions. The actions that an organization works to accomplish are determined by the organization’s goals. The steps which are followed to accomplish actions are defined as the organization’s process. The fact that organizations are comprised of groups of people means that organizations are social systems. Therefore, the key components in the organization are the individual employee and the collection of individual employees who work as a team. The methods and techniques used to organize the individual employees into teams forms the basis of the organization’s structure. The organization’s structure and process form the foundation for how the organization develops products and provides services.

The overall desire of organizations to accomplish a goal means that organizations need to create strategies and plans to define goals as well as to achieve those goals. Defining the goals begins at the executive level with strategic objectives for the
organization and works down to specific goals for a team as well as personal goals for each individual employee. The definition of goals is the first step required in order to execute the plan to achieve the goals. The goals are refined and broken down into objectives and then the plan to achieve each objective is created. This planning involves following a defined process in order to successfully achieve the goal. This process includes defining the goals, setting objectives to reach the goals, creating processes to achieve the objectives and identifying the resources necessary to follow the process in order to achieve the objectives and goals. (McNamara, 2007). One of the foundations of an organization’s structure depends on the organization’s team structure and goals.

3.2 Overview

There are several different types of organizational structures and configurations and many organizations use a hybrid organizational structure which is an amalgam of more than one organizational structure. The following is a general overview of various organizational structures. One of the main categorizations of organizations is based on the bureaucracy of the organization. The organization can be pre-bureaucratic, which is generally a smaller organization at the entrepreneurial stage which has not yet standardized its processes. Pre-bureaucratic organizations tend to have a strong centralized power structure with a traditional domination or charismatic domination authority (Weber, 1978). A bureaucratic organization is one that has instituted standardization with its processes. Bureaucratic organizations tend to be larger organizations that are more mature. Bureaucratic organizations can be structured by
function, division, or in a matrix style. A post-bureaucratic organization is in contrast with the bureaucratic organization and is based more on consensus then on hierarchy. Heckshcer and Donellson describe these types of organizations as “cleaned up bureaucracies” rather than non-bureaucratic. In general, most organizations involved in software development are pre-bureaucratic or bureaucratic. Due to the start-up nature of software development there are potentially more pre-bureaucratic then bureaucratic software development organizations.

Another type of organizational classification is into functional, project, and matrix organizations. Organizations that have these structures can be pre-bureaucratic in nature but are more likely to be bureaucratic. A functional organization is organized by functional units with managers representing those units. For example, an organization may have a Development Manager, an Operations Manager, and a QA Manager each managing these specific functions of the organization. Functional managers have their own teams and they manage the scheduling and resourcing for those teams. There is no single one of these managers who has responsibility for the project. In these organizations there are generally rules and processes that define how the functional areas interact – the more bureaucratic the organization the more rules there will be.

A project based organization is organized by project and the associated project teams. In these organizations employees are affiliated with a functional area but are project members and work full time on the project team. The functional managers
only oversee the human resources aspect of the employee such as performance reviews, vacation requests, and professional development. Functional managers do not manage the employee’s day to day work. In the project organization the work is managed by project managers and all members of the team essentially report to the project manager for their tasks. The project managers manage a cross functional team with members representing many different functional areas. The project managers are often under the direction of an organizational project manager or program manager. These organizations are also referred to as PMOs or Project Management Offices.

A matrix organization combines both the functional and project organizational structures. There are several different techniques for combining these structures which include light weight and heavy weight structures. A light weight matrix organization is mostly functionally structured however a project manager is added to coordinate the projects, solve conflicts between functional units, and work at achieving project goals. The project manager is generally a peer of the functional managers and often reports to the same manager as the functional managers. The functional managers guide the work but with the guidance of the project manager. In a heavy weight matrix structure the project manager has greater power and responsibility and is often a senior manager or director. Under this system the project manager guides more of the work and it is possible that the functional managers may report to the project manager.

Organizations can also be structured based on the complexity of the products developed and the amount of effort required to create the product. These two concepts
are often considered in combination in terms of product complexity combined with work effort complexity. The first combination is a low complexity and low work effort. This is usually a functionally structured environment with a small development department. In these organizations the development effort is often in support of the organization or of the process that produces the main organizational product. These organizations do not generally produce software as their main product. The second combination is low complexity but high work effort. This is usually a functionally structured environment or a light weight matrix structure. The third combination is a complex product with low work effort. This type of combination usually occurs in a matrix structure. In this type of organization, with this type of combination there is usually a high level of risk to projects due to project prioritization conflicts, fluctuations in workload, and conflicts over resource allocation. The fourth combination is a complex product with high work effort. In this type of environment a project based environment works best as the complexity of the project and the amount of work required necessitate the direction of a project manager.

Organizations can be categorized according to the tasks they perform. For example it is possible to classify organizations according to those who perform custom work or custom production, those that create large volumes of one or more basic items, and those that have continuous production. (Borgatti, 1996). Organizations that perform custom work often have very direct customer involvement and much of the work is directly customer driven. Most software companies fall into the custom production or large volume categories – depending on if they are doing
custom development work or selling commercial off-the-shelf software. In custom production companies workers have specialized knowledge and their knowledge is important. These types of organizations are often less hierarchical, have fewer managers, have unpredictable work processes, and possess very organic structures. Large volume organizations tend to have larger and deeper hierarchies, higher numbers of managers, and a very bureaucratic structure. As a subset of the types of tasks organizations can also be categorized by size, including the number of employees, the number of customers and the amount of resources. Often the management practices which are followed are directly related to the size of the organization.

Another organizational characteristic is the organizational change environment as described by Frank Kand (Kand, 1998). Kand describes the types of organizational change environments as unchanging, turbulent, uncertain, and adaptive and provides details as follows:

- **Unchanging.** In this environment requirements never or rarely change and they can always be fully stated and comprehended at the start of the project. Unchanging environments are very concerned with accuracy. Generally these environments are bureaucratic with a formal development process – often a waterfall or spiral process.

- **Turbulent.** In this environment the requirements are under constant change and are not well understood at the start of the project. These organizations generally use an iterative technique such as rapid application development.
• Uncertain. In this environment the requirements are unknown. Experimental process models that use prototyping and RAD work best. Prototyping methods can be especially beneficial as they allow a vision of the product and can help in determining the requirements.

• Adaptive. In this environment the environment itself changes and causes requirements to change. This organization would also generally use a technique such as rapid application development although the situation would likely be in more flux due to both the organization and the requirements being changeable.

The change environment is a very important feature to analyze when considering which development method to use as the basis of many of these methodologies is how the methodology deals with requirements and changes to requirements.

Some organizational theorists have worked to combine several organizational characteristics into basic organizational types. In his book *Mintzberg on Management*, Henry Mintzberg describes the basic organizational types as entrepreneurial, machine, professional, diversified, innovative, missionary and political. The details regarding these of these types are as follows:

• Entrepreneurial. This is a simple and informal organizational type. It is designed to be flexible and has very little hierarchy. In most cases the chief executive controls the organization directly. These organizations have simple and dynamic environments and are often operating in startup and crisis mode.
Most of these organizations are highly responsive and have a strong sense of mission. The potential problem with these organizations is that they can be vulnerable and restrictive and are in danger of imbalance.

- **Machine.** These organizations are more complex and formal. They are hierarchical in nature and are characterized by a centralized bureaucracy and established, formal processes. These organizations have a strong division of labor where employees are generally divided into functional groupings. The environments are simple and stable and the organization is more mature and resistant to strategic change. The organization is generally efficient, reliable, precise, and consistent. These organizations can have problems with a control obsession which leads to human problems, coordination problems, and adaptation problems.

- **Professional.** This organization is bureaucratic and decentralized and has minimal hierarchy. Generally there are wide spans of control and the organization is complex and stable. Generally these organizations are comprised of groups of individual professionals working autonomously. The overall strategy is stable but the specific details are often changing. These organizations are focused on democracy and personal autonomy. These organizations can have problems with coordination and are reluctant to innovate.

- **Diversified.** This organization is characterized by autonomous market divisions under a central administration. These organizations offer diverse products and services and are generally larger and more mature. Diversified
organizations have a portfolio of businesses which spreads risk but can be costly and discourage innovation. These organizations may encourage irresponsible behavior and may cause individuals to set non-measurable goals.

- **Innovative.** This organization is fluid and organic and is generally based on a decentralized structure. The organization is usually comprised of functional experts and relies on mutual coordination. These are commonly a matrix structure in a complex and dynamic environment which undergoes frequent change. The organization is usually young and is going through a start-up process. Many software development companies can be classified in this category.

- **Missionary.** This organization is based around a strong system of values and beliefs and is based on a sense of mission. They often have charismatic leadership and manage the organization through selection and indoctrination of members. The organization is basically decentralized but has powerful centralized controls.

- **Political.** This is not so much an organization but a characteristic that can appear in different organizations. There are issues with self interested power, conflict, and political games. The political aspects are usually overlaid on a conventional organization and are characterized by a play of informal power and a development of forms through conflict development and resolution.
The organizational types most often associated with software development organizations are Entrepreneurial, Diversified and Innovative. Software development projects may also exist at organizations that are Machine and Professional types.

Given the various definitions and categories described above, organizations can generally be analyzed as having one of each of the following characteristics:

- Bureaucratic or Non-Bureaucratic
- Functional structure or project structure
- Low complexity projects or high complexity projects
- Low work effort projects or high work effort projects
- Custom work or large volume work
- Unchanging or Changing
- Large or Small

Once an organization has been analyzed based on these characteristics it can be associated with an organizational type. An overview of the organizational types is provided in the table below. This table can be used to determine an organization’s type by selecting the organizations characteristics in the table and matching them to the appropriate structure.
### Table 1: Organizational Characteristics and Type

<table>
<thead>
<tr>
<th></th>
<th>Entrepreneurial</th>
<th>Innovative</th>
<th>Machine</th>
<th>Diversified</th>
<th>Professional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bureaucracy</strong></td>
<td>Non-Bureaucratic</td>
<td>Non-bureaucratic</td>
<td>Bureaucratic</td>
<td>Bureaucratic or Non-Bureaucratic</td>
<td>Bureaucratic</td>
</tr>
<tr>
<td><strong>Structure</strong></td>
<td>Matrix (with less hierarchy)</td>
<td>Matrix</td>
<td>Functional</td>
<td>Functional</td>
<td>Functional</td>
</tr>
<tr>
<td><strong>Complexity</strong></td>
<td>High</td>
<td>High</td>
<td>Low or High</td>
<td>Low or High</td>
<td>High</td>
</tr>
<tr>
<td><strong>Environment</strong></td>
<td>-Dynamic</td>
<td>-Formal</td>
<td>-Decentralized</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Work effort</strong></td>
<td>High</td>
<td>High</td>
<td>Low or High</td>
<td>Low or High Custom or Large Volume</td>
<td>Low or High Custom</td>
</tr>
<tr>
<td><strong>Work Type</strong></td>
<td>Custom</td>
<td>Custom</td>
<td>Large volume</td>
<td>Custom or Large Volume</td>
<td></td>
</tr>
<tr>
<td><strong>Change</strong></td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>Low or High</td>
<td>Low or High</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>Small/Medium</td>
<td>Small/Medium</td>
<td>Medium/Large</td>
<td>Medium/Large</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Flexibility</strong></td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Problems</strong></td>
<td>-Vulnerable</td>
<td>-No standardization</td>
<td>-Control obsession</td>
<td>-Cohesive strategies or processes</td>
<td>-unable to produce new products or services</td>
</tr>
<tr>
<td></td>
<td>-Restrictive</td>
<td>-Low unity of command</td>
<td>-Human problems</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Imbalanced</td>
<td>-Coordination problems</td>
<td>-Coordination problems</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td>-Strong sense of mission</td>
<td>-Teamwork</td>
<td>-Efficient</td>
<td>-Autonomous units under a central structure</td>
<td>-Independent workers</td>
</tr>
<tr>
<td></td>
<td>-Responsive</td>
<td>-Many middle managers</td>
<td>-Reliable</td>
<td>-Professional standards</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-Precise</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-Consistent</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An organization can use the table above to determine – at a very basic level – their organizational structure. For example, a small company which performs a great deal of custom work falls into the entrepreneurial or innovative categories. Entrepreneurial organizations tend to have less hierarchy in the organizational structure. They are also more likely to be working in a startup or crisis mode. Therefore, an organization could be classified as entrepreneurial if there is a very flat hierarchy or innovative if there is more hierarchical structure.
Chapter Four - Selecting a Software Development Methodology

3.3 Selecting a Methodology

One of the most important decisions when developing software is selecting an appropriate software development methodology. Selecting the appropriate methodology can increase the probability of project success. It is very important that an organization perform the research on all the methodologies they may select prior to starting the development process and make a decision regarding what methodology to select. Some time should be allocated for research and decision making as well as for training and implementation. Organizations that are currently using a specific software development methodology should take care when changing methodologies as:

“Itering through several different approaches that do not work, until you find one that does, hurts credibility and morale, as well as being costly.” (Norman, 2005). The act of selecting a new development methodology has a large impact on the organization and is a task that should be performed carefully and with due consideration to both tactical and strategic planning. It is also important to consider the impact that the selection would have on the employees of the organization. In order to have an effective transition to a new methodology it is beneficial to involve all stakeholders in the company in the decision of what methodology to use and ensure that communication processes are in place to ensure that any organizational changes are well communicated to the organization.
Selecting the appropriate software development methodology should be managed as a discrete project which is completed prior to any other projects using the new methodology. This project should follow the basic processes in project management including gathering requirements, creating a design, implementing the design, and testing the implementation. One of the key steps in the project to select a development methodology is to determine all of the stakeholders and ensure that all of the stakeholders are identified and included in any communication related to the selection of a software development methodology. The stakeholders may be customers outside your organization as well as executive, management, project teams, and shareholders. One of the most important stakeholders is senior management and other senior personnel as senior management buy in is usually critical to project success. Any new process will need the support of senior management therefore it is important that this group in particular be involved in the process. It is also important that all the employees affected by this change are included in information surrounding the change. This will make it easier for them to accept any new changes that are implemented.

It is important to take some time up front to perform a complete analysis of the organization prior to starting the project to select a development methodology. One useful endeavor is to ensure that you have compiled a document which lists the methodologies that have been tried in the past and what the success was with those methodologies. It is useful to examine both the methodologies that were accepted and are in use as well as those that were attempted and failed. It is also useful to determine
why these methodologies failed. This will provide greater insight into the organization as well as ensuring that you have a good understanding of the history that the team has with various methodologies. This can prevent issues with team morale and resistance to change which can occur if the new methodology being suggested has already been attempted and failed. It can be very useful to run a pilot project with a new methodology on a smaller and less critical project. For this initial implementation ensure that there is additional contingency built into the schedule.

One of the most important aspects to consider when selecting a software development methodology is the organization size. There is a large difference between small and large organizations in terms of the type of methodology that is most appropriate. Large organizations have many layers of management and approval and often require several different approvals for a new process. In many cases there is a requirement for a decision by committee. There are many different sizes of projects at a larger company – but in general there are more projects of a medium and large size. Larger organizations have many different stakeholders who must be identified and kept informed. At a larger organization there can be more technically sophisticated contacts who want to have input into the process. In order to gain approvals for money there are generally many different approvals necessary and complicated funding models. The tolerance for risk at a larger organization is generally lower. Also, there can be more political considerations related to projects and projects may be based on political justifications over financial justifications. (Norman, 2005). Small organizations have fewer decision makers but more power per decision maker. They
generally have functionality rich, feature light, and low cost projects. Smaller companies have less technically sophisticated contacts and can support higher risk tolerance. However, with smaller companies there is a risk that one or two unsuccessfully projects can break the company. In many cases, at smaller companies, team members perform multiple roles. (Norman, 2005)

Each project has custom characteristics that affect which software development methodology is best to use. The most important is project size in terms of the scope of the project. Another important consideration is the number of projects that are going through the organization at any given time. This is tied to the characteristics of the project team. It is important to understand the resources available to each project and the skills of those resources. It is also useful to determine if the people on the project team are dedicated to the project or if they are performing duties on multiple projects and/or working on a functional team as well. The experience of the team members is important but in particular it is important to note the characteristics of the project manager and how the project manager fits into the hierarchical structure. It is particularly important to understand how much experience the project manager has and if a project manager can be dedicated to the project to plan, coordinate, and take care of the project communication. Another key aspect of the project is analyzing who the customer is for the project and determining if they are internal customers, external customers or both. Once it is understood who the customers are it is useful to analyze their characteristics and priorities. Part of this is examining how much interaction the customer would like to have with the team throughout the project development cycle.
3.4 Organizational Strategies

There are several strategies that organizations can use when selecting a software development methodology. After the organization and the project types have been analyzed a general methodology can be selected. The basic compatibility between organizational types and software development methodologies is presented in the table below. This table presents the organizational types along the top row and software methodologies along the first column. The suggestions in the table above are just general guidelines. As discussed there are other considerations that will assist in the selection of a software development methodology including the type of customer, the ability of the organization to handle change, and the type and number of employees.

Table 2: Methodologies and Organizational Structures

<table>
<thead>
<tr>
<th></th>
<th>Entrepreneurial</th>
<th>Innovative</th>
<th>Machine</th>
<th>Diversified</th>
<th>Professional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Spiral</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Prototype</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RAD</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Agile</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>XP</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Formal</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMM/CMMI</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ISO</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Six Sigma</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ad-Hoc</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

For example, many software development organizations fall under the entrepreneurial structure. For an entrepreneurial organization the recommended methodologies are Spiral, prototype, RAD, Agile, and XP. Entrepreneurial organizations have a high degree of change and require a methodology which supports change. The work of an entrepreneurial organization tends to be high effort and high
complexity and generally is custom work. The iterative methodologies work best in environments where there is a great deal of change – while the waterfall, formal methods, MDA, CMMI, ISO, Six Sigma and Ad hoc methods will not work as well in environments with high change. At a high level the organization can determine that iterative methods will work best. They can also avoid using methodologies that will likely not be very successful.

Another very useful technique is to combine methods and models. Many organizations have characteristics which would work well with part of a certain methodology but would also work well with another methodology. It is possible to take parts of compatible methodologies and use them jointly on a single project. One of the more effective techniques is to combine certain components of different methodologies and models. This can be done by agreeing to combine models for the basic process to be followed for a project as well as changing models if the project situation changes. Often organizations are too concerned about selecting the single process methodology that they will use to work with all their projects. It is unlikely that such a methodology exists as every organization has different projects which require different methods and processes. At the same organization different technologies are used which require different techniques and methodologies. Every individual and team is unique and the unique needs of both teams and individuals must be considered. Each customer is different and the customer requirements have a large impact on the type of methodology used. The following table provides a guideline on which methods are compatible with each other and can be combined:
When an organization is looking at software development processes and methodologies there are also a few very general suggestions. As a first step, work through the tables above and eliminate any methodologies that will not work with the organization. This will leave a subset of methodologies that could work. At this point examine the project and determine which of these methodologies are a good fit for the project and eliminate any that are not a good fit. Once this step is complete examine the project’s customer and review the remaining methodologies. Eliminate any that are not a good fit for that specific customer. At this point there should be one or more methodologies remaining. These can then be examined in greater detail and either one methodology selected or different components of the remaining methodologies selected. Care must be taken at this point to ensure that if components of multiple methodologies are being used that they are not in conflict with each other. Scott Ambler suggests the following when selecting a methodology (Ambler, 2006):

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall</td>
<td>Spiral, Prototype, Agile, XP, Formal Methods</td>
</tr>
<tr>
<td>Spiral</td>
<td>Prototype, RAD, Agile, XP, Formal Methods</td>
</tr>
<tr>
<td>Prototype</td>
<td>Prototype, RAD, Agile, XP, Formal Methods</td>
</tr>
<tr>
<td>RAD</td>
<td>Prototype, RAD, Agile, XP, Formal Methods</td>
</tr>
<tr>
<td>Agile</td>
<td>Prototype, RAD, Agile, XP, Formal Methods</td>
</tr>
<tr>
<td>XP</td>
<td>Prototype, RAD, Agile, XP, Formal Methods</td>
</tr>
<tr>
<td>Formal Methods</td>
<td>Waterfall, Spiral, Prototype, Agile, XP</td>
</tr>
<tr>
<td>CMM/CMMI</td>
<td>Prototype, RAD, Agile, XP, Formal Methods</td>
</tr>
<tr>
<td>ISO</td>
<td>Prototype, RAD, Agile, XP, Formal Methods</td>
</tr>
<tr>
<td>Six Sigma</td>
<td>Prototype, RAD, Agile, XP, Formal Methods</td>
</tr>
<tr>
<td>Ad-Hoc</td>
<td>None</td>
</tr>
</tbody>
</table>
- Select a subset of the methods available that are the best fit for the organization. Ensure that these methods are generally compatible with each other.

- Customize the process to the project team that will be working on that specific project.

- Define the specific goals for the project. Do not focus on the items and deliverables that are specified in the methodology as deliverables.

- Define, document, and communicate the terminology and the process to all stakeholders. Ensure that there is stakeholder agreement.

- Ensure that those developing the process to be used by the organization are flexible to ideas and change.

- The primary focus should be on prioritization.

Once these steps are followed the organization should have a candidate methodology selected.
4. Chapter Five - Implementing a Software Development Methodology

Chapter five provides an overview of how the software methodologies can be implemented at an organization. This chapter will also present some areas for future research into the area of software development methodologies.

4.1 Implementing a new Software Development Methodology

Once the organization has selected a candidate Software Development Methodology it is time to plan for the implementation of that methodology. It is very important to remember that the initial implementation of the methodology will require a substantial amount of additional time and effort. An essential strategy in this implementation is to manage the implementation of the new methodology as its own project. Before this project starts ensure that there is buy in from all stakeholders and particularly from senior staff and senior management. Also, ensure that there is information and documentation available for anyone who is interested in the project. In terms of the project to implement the new methodology - ensure that any project estimates include adequate time to implement the new methodology. It is helpful to place substantial contingency time in this schedule. Before initiating any work on the project to bring in the new methodology ensure that all of the project planning is complete, documented, and has stakeholder buy-in. It is very useful to run information
sessions for all of the employees who will be affected by the change in order to reduce resistance to implementing the new methodology.

As a milestone in the project to bring in the new methodology it can be very useful to organize and complete a small pilot project using the new process. The pilot project should be low risk but should be a good example of the type of project that the organization will be working on. This could be a project to implement a small, discrete module which is part of the larger project or perhaps a utility that is used with the system. It is helpful at this point to involve high level stakeholders to go through the experience of using the new methodology. It is also important to select this project carefully so that it does not have impact on work that is already scheduled. Ensure that the project to implement the new methodology has a strong project manager and contains manageable and measurable milestones. This will allow for progress to be easily measured and communicated to the larger group. Before the project is underway have a general brainstorming session and define the risks for the project. In particular question stakeholders on what they perceive the risks are with the new methodology. Each risk should be recorded and tracked carefully. Once the project is complete a full post-mortem evaluation should be done which includes everyone involved in the project. This post-mortem should produce clear recommendations for improvement which should be implemented prior to using the new process on another project.

If there are any stakeholder concerns regarding the implementation of the new methodology ensure that the benefits of the new methodology have been
communicated. Work to research the bottom line financial benefits of the new methodology and communicate these to the group. If you are able to tie the project to financial benefit you will more easily achieve buy-in from the influential senior management. Financial benefits can be shown by performing a cost benefit analysis with the new methodology or creating a SWOT chart. It is important not to get too involved in the details and to demonstrate how this new methodology will benefit the employees, the team, and the organization as a whole. At this point it can be useful to review problems and issues that the team has been facing in the past and demonstrate how the new methodology will assist in resolving these issues.

One very important consideration is that the team should be open to changing the methodology if necessary. There should be regular reviews of the progress and if there are issues these should be identified and addressed. The overall goal should be the continuous improvement of the process. This may involve small changes to the process or even larger changes such as using a different methodology for certain projects. As part of the process implementation plan there should be a formal change management process established. This process should have a mechanism for requesting feedback from the employees involved in working with the new process. The stakeholders should be fully educated both on the possibility that the methodology may change as part of this project and what the process for change will be.
An essential component to the success of implementing a new process is to ensure that it is well understood. To accomplish this goal, the process should be fully documented prior to being implemented. It should be clear to all the stakeholders when they need to complete documents as part of the process and where they will find and store these documents. In order to make it simpler for the stakeholders it is best to create and use very basic templates to ensure that the process is well understood. All of the documents and templates should be documented themselves and the stakeholders should be educated regarding how to use them. It is also helpful to run through a training program to educate employees regarding the new process. In terms of project status it is helpful to validate that there is a simple method for reporting project status to ensure that all stakeholders are kept informed and that the success of the methodology is evaluated.

One very important thing to remember is that managing the introduction of a new methodology into an organization is managing change to that organization. Any methodology that is introduced to the organization will be new to the members of the organization and will face the challenges that occur whenever new technology is introduced. Some of these challenges can be faced by working on communication, documentation, and user education. If the organization is very resistant to change or there are other issues to consider it may be beneficial to bring in an organizational change specialist. Alistair Cockburn has some valuable advice on the philosophy of methodology selection. He indicates that the selection of a methodology should be considered as a “‘try this’ instead of ‘do this’” (Cockburn, 1997). It is really only in
the practice of using a new process on its own software development projects that an organization will be able to determine if that methodology is a good fit.
5. Chapter Six - Conclusion

5.1 Conclusions

This paper provides an overview of both software development methodologies and organizational structures and provides guidance on how to select a software development methodology for a project based on characteristics of the organization’s structure. Selecting the correct software development methodology for a project can help projects to release successfully, on time, and within budget. Once an organization has determined which methodologies will work best for its projects it can ensure that there is a repeatable process established that will ensure successful projects. The goal of this paper has been to provide enough information to allow an organization to select the software development methodology that is most likely to be the best fit for their organization and projects. Although the methodologies are not presented in a great deal of detail the goal has been to present enough information to allow for a selection to be made and additional research into the methodology to be performed as necessary.

5.2 Recommendations for Further Research

There is potential for additional research into additional software development methodologies – particularly investigating several of the methodologies in detail. There is also additional work to be done in designing a more detailed methodology selection chart which incorporates factors other than organizational structure. An interesting area of research would be the analysis of different types of projects and
which development methodologies would work best for specific types of projects.

Another interesting research project would be to run a trial whereby the suggestions of
this paper are implemented on an actual software project as a trial.
6. References


Adam Kolawa, *Which Development Method is right for your Project?*, 2007, Retrieved October 8, 2007 from


Winston Royce, “Managing the Development of Large Software Systems”,

*Proceedings of the 9th international conference on Software Engineering*,


Software Engineering Institute, “Cleanroom Software Engineering”, 2007, Retrieved on October 29, 2007 from

http://www.sei.cmu.edu/str/descriptions/cleanroom.html

Software Engineering Institute, *Capability Maturity Model Integration Version 1.2 Overview Presentation*, 2007, Retrieved on October 31, 2007 from


Retrieved on October 31, 2007 from

http://sepo.spawar.navy.mil/12207_SPIWG.ppt
