A Comparison between Agile and Traditional Software Development Methodologies

M. A. Awad

This report is submitted as partial fulfilment
of the requirements for the Honours Programme of the
School of Computer Science and software Engineering,
The University of Western Australia,
2005

Abstract

Software has been part of modern society for more than 50 years. There are several software development methodologies in use today. Some companies have their own customized methodology for developing their software but the majority speaks about two kinds of methodologies: heavyweight and lightweight. Heavyweight methodologies, also considered as the traditional way to develop software, claim their support to comprehensive planning, detailed documentation, and expansive design. The lightweight methodologies, also known as agile modeling, have gained significant attention from the software engineering community in the last few years. Unlike traditional methods, agile methodologies employ short iterative cycles, and rely on tacit knowledge within a team as opposed to documentation.

In this dissertation, I have described the characteristics of some traditional and agile methodologies that are widely used in software development. I have also discussed the strengths and weakness between the two opposing methodologies and provided the challenges associated with implementing agile processes in the software industry. This anecdotal evidence is rising regarding the effectiveness of agile methodologies in certain environments; but there have not been much collection and analysis of empirical evidence for agile projects. However, to support my dissertation I conducted a questionnaire, soliciting feedback from software industry practitioners to evaluate which methodology has a better success rate for different sizes of software development. According to our findings agile methodologies can provide good benefits for small scaled and medium scaled projects but for large scaled projects traditional methods seem dominant.

Keywords: Software Process, Software Development Methodology, Agile, Heavyweight
CR Categories: D.2.1 [Requirements/Specifications] Methodologies, D.2.9 [Management] Life cycle, K.6.3 [Software Management] Software Process, K.6.3 [Software Management] Software Development

Acknowledgements

Firstly I would like to thank my family for their support, encouragement and love that helped me get through University. This thesis would not have been possible without them and I hope I have made them proud.

My supervisors, Alex Reid and Terry Woodings, for all the help I received over the research project. Had not been for your guidance and experience I would not have been able to continue. For without their guidance, support and assurance that everything is going fine I would have had a stress attack. Terry Woodings, for providing a positive outlook on my progress when I felt that there was no progress and great feedback when I didn't get something as he always knew generally everything about anything. Alex Reid, for taking me as his student and giving me confidence that I could make this thesis into a really good one. I really learnt a lot more than just about my thesis. Thanks!

My sister and cousins for their hospitality while my parents were not here. Without them making me stay at home on weekends and not out partying I would have been way behind in my work.

Finally, thank you to all my friends for their moral support and encouragement throughout the last semester.

Contents

List of Figures

List of Tables

1.0 Introduction

Software has been part of modern society for more than 50 years. Software development started off as a messy activity often mentioned as “code and fix”. The software was written without much of a plan, and the design of the system was determined from many short term decisions. This worked well for small systems but as systems grew it became more difficult to add new features and bugs were harder to fix. This style of development was used for many years until an alternative was introduced: Methodology. Methodologies impose a disciplined process upon software development with the aim of making software development more predictable and more efficient.

Traditional methodologies are plan driven in which work begins with the elicitation and documentation of a complete set of requirements, followed by architectural and high level design development and inspection. Due to these heavy aspects, this methodology became to be known as heavyweight. Some practitioners found this process centric view to software development frustrating and pose difficulties when change rates are still relatively low. As a result, several consultants have independently developed methodologies and practices to embrace and respond to the inevitable change they were experiencing. These methodologies and practices are based on iterative enhancements, a technique that was introduced in 1975 and that has become known as agile methodologies.

The name “agile” came about in 2001, when seventeen process methodologists held a meeting to discuss future trends in software development. They noticed that their methods had many characteristics in common so they decided to name these processes agile, meaning it is both light and sufficient. In consequence to this meeting, the “Agile Alliance” and its manifesto for agile software development emerged. The agile methods claim to place more emphasis on people, interaction, working software, customer collaboration, and change, rather than on processes, tools, contracts and plans

Agile methodologies are gaining popularity in industry although they compromise a mix of accepted and controversial software engineering practices. The software industry would most likely find that specific project characteristic such as objective, scope, requirements, resources, architecture and size will determine which methodology suits them best. Either agile or heavyweight or maybe a hybrid of the two. In the past few years, anecdotal evidence and success stories from practicing professionals suggests that agile methods are effective and suitable for many situations and environments. However, empirical studies are urgently needed for evaluating the effectiveness and the possibilities of using agile software development methods.

In today’s increasing volatility and uncertainty, talented people want to work in an organization in which they have more control over how they work and how they interact with peers, customers and management. Problems are changing, people are changing and ideas are changing. While there is still a need for plan driven style development and

management in some situations the bigger growth lies in agile and flexible. This report investigates heavyweight and agile methodologies for their suitability in software development and review anecdotal data given from practitioners to determine which methodology suits best.

1.1 Overview

Our goal, therefore, is to begin filling in the gap of methodologies by conducting a detailed review of both heavyweight and agile methodologies. For heavyweight method, I reviewed several methods such as Waterfall, Unified Process and Spiral. I further discussed an overall view of the characteristics of heavyweight methods. Next, I followed the same procedure for agile methodologies. I introduced some agile approaches such as Extreme Programming, Scrum, Dynamic System Development Method, Feature Driven Development and Adaptive Software Development underlining the characteristics of agile methods. Furthermore, I carried out a comparison of the different agile methods in order to highlight the similarities and differences between them. The next section criticizes the limitations of each heavyweight and agile methods. Following this, the challenges associated with implementation of agile processes in the software industry according to software practitioners and anecdotal evidence. To conclude, I conducted a questionnaire to gather feedback from software developers in Perth and analyzed which methodology was used to develop software and as well as their views on agile and heavyweight methodologies.

2.0 Heavyweight Methodologies

Heavyweight methodologies are considered to be the traditional way of developing software. These methodologies are based on a sequential series of steps, such as requirements definition, solution building, testing and deployment. Heavyweight methodologies require defining and documenting a stable set of requirements at the beginning of a project. There are many different heavyweight methodologies but I will limit our discussion to the three most significant methodologies: Waterfall, Spiral Model and Unified Process.

2.1 Waterfall

During the 1960s, "code and fix" was the method employed by software developers. As Christophe Thibuat describes, "one year of slamming code, one year of debugging". Due to this difficult nature of "code and fix" approach, Winston Royce in 1970 proposed the waterfall methodology. The waterfall approach emphasizes a structured progression between defined phases. Each phase consists on a definite set of activities and deliverables that must be accomplished before the following phase can begin. The phases are always named differently but the basic idea is that the first phase tries to capture *What* the system will do, its system and software requirements, the second phase determines *How* it will be designed. The third stage is where the developers start writing the code, the fourth phase is the *Testing* of the system and the final phase is focused on Implementation tasks such as training and heavy documentation. However, in engineering practice, the term waterfall is used as a generic name to all sequential software engineering methodology. Figure 1 below shows a traditional waterfall lifecycle and Figure 2 illustrates the deliverables needed for each phase to be able to proceed to the next.

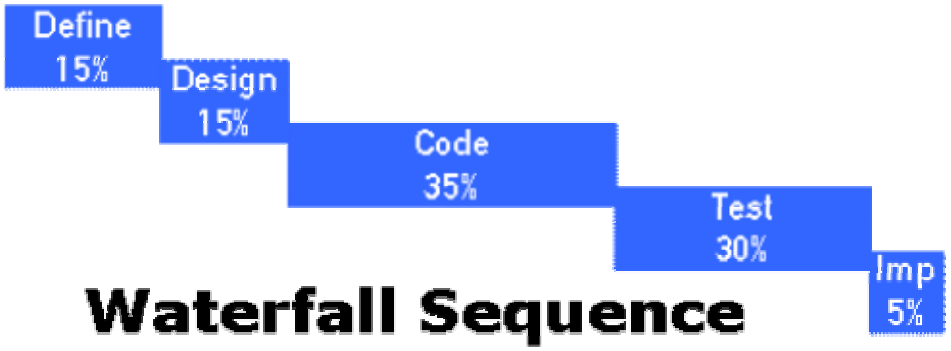


Figure 1: Waterfall Lifecycle [1]

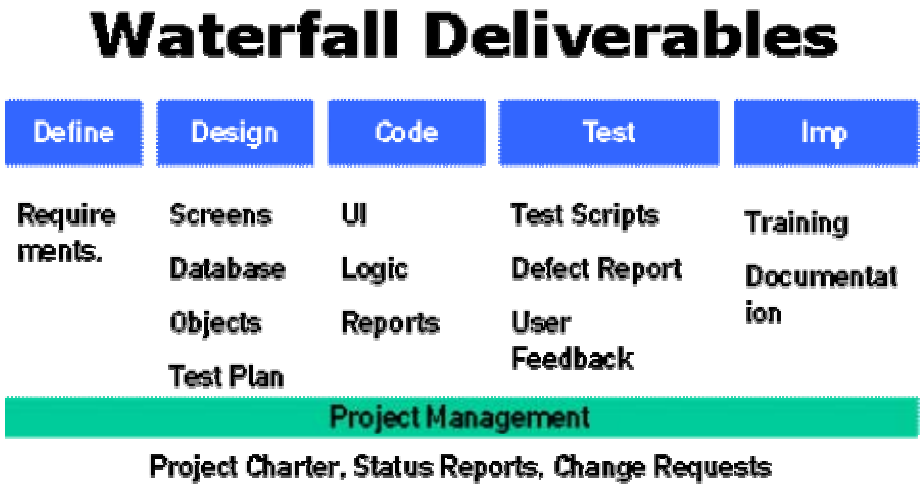


Figure 2: Waterfall Deliverables [1]

2.2 Unified Process

All efforts, including modeling, is organized into workflows in the Unified Process (UP) and is performed in an iterative and incremental manner. The lifecycle of the UP is presented in Figure 3. Some of the key features of the UP are as follows [2]:

- It uses a component based architecture which creates a system that is easily extensible, promotes software reuse and intuitively understandable. The component commonly being used to coordinate object oriented programming projects.
- Uses visually modeling software such as UML – which represent its code as a diagrammatic notation to allow less technically competent individuals who may have a better understanding of the problem to have a greater input.
- Manage requirements using use-cases and scenarios have been found to be very effective at both capturing functional requirements and help in keeping sight of the anticipated behaviors of the system.
- Design is iterative and incremental – this helps reduce project risk profile, allows greater customer feedback and help developers stay focused.
- Verifying software quality is very important in a software project. UP assists in planning quality control and assessment built into the entire process involving all member of the team.

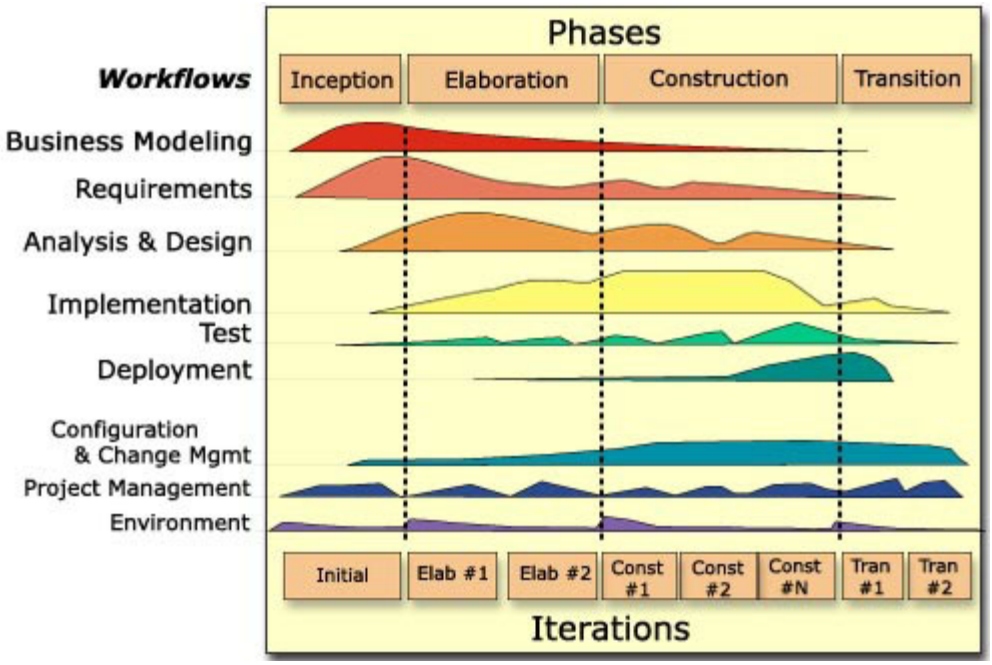


Figure 3: Unified Process Lifecycle [2]

These key features above are guidelines to be adhered throughout a projects' lifecycle. To determine the length of the project, UP divides the project into four phases which are shown above in Figure 3 and discussed below [3]:

- Inception – By the end of this process a business case should have been made; feasibility of the project assessed; and the scope of the design should be set.
- Elaboration – In this phase a basic architecture should have been produced and a plan of construction agreed. Furthermore, a risk analysis takes place and those risks considered to be major should have been addressed.
- Construction – This process produces a beta-release system. A working system should be available and sufficient enough for preliminary testing under realistic conditions.
- Transition – The system is introduced to the stakeholders and intended users. It is crossed when the project team and the stakeholders agree that the objectives agreed in the inception phase have been met and the user is satisfied.

There are approximately 50 work products to be completed in UP [4].All this documentation and this rigid approach adds a lot of complexity to UP. As well, UP predefines roles to the project team making it less flexible.

2.3 Spiral Model

Another heavyweight software development model is the spiral model, which combines elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. The spiral model was defined by Barry Boehm, based on experience with various refinements of the waterfall model as applied to large software projects. There are four main phases of the spiral model [5]:

- Objective setting – Specific objectives for the project phase are identified
- Risk assessment and reduction – Key risks are identified, analyzed and information is obtained to reduce these risks
- Development and Validation – An appropriate model is chosen for the next phase of development.
- Planning – The project is reviewed and plans are drawn up for the next round of spiral

2.4 Heavyweight Characteristics

Heavyweight methodologies have been around for a very long time. They impose a disciplined process upon software development with the aim of making software development more predictable and more efficient. They have not been noted to be very successful and are even less noted for being popular. Fowler criticizes that these methodologies are bureaucratic, that there is so much to follow the methodology that the whole pace of development slows down [6]. The heavyweight methodologies have these similar characteristics.

Predictive approach – Heavyweight methodologies have a tendency to first plan out a large part of the software process in great detail for a long span of time. This approach follows an engineering discipline where the development is predictive and repeatable. A lot of emphasis is put on the drawings focusing on the need of the system and how to resolve those needs efficiently. The drawings are then handed over to another group who are responsible for building the system. It is predicted that the building process will follow the drawings. The drawings specify how they need to build the system; it acts as the foundation to the construction process. As well, the plan predicts the task delegation for the construction team and reasonably predicts the schedule and budget for construction.

Comprehensive Documentation – Traditional software development view the requirements document as the key piece of documentation. A main process in heavyweight methodologies is the big design upfront (BDUF) process, in which a belief that it is possible to gather all of a customer's requirements, upfront, prior to writing any code. Again this approach is a success in engineering disciplines which makes it attractive to the software industry. To gather all the requirements, get a sign off from the customer and then order the procedures (more documentation) to limit and control all

changes does give the project a limit of predictability. Predictability is very important in software projects that are life critical.

Process Oriented - The goal of heavyweight methodologies is to define a process that will work well for whoever happens to be using it [6]. The process would consist of certain tasks that must be performed by the managers, designers, coders, testers etc. For each of these tasks there is a well defined procedure.

Tool Oriented – Project management tools, Code editors, compilers, etc. must be in use for completion and delivery of each task.

3.0 Agile Modeling

Agile – devoting "the quality of being agile; readiness for motion; nimbleness, activity, dexterity in motion" as mentioned in the Oxford Dictionary [7] – software development methods are attempting to offer once again an answer to the eager business community asking for lighter weight along with faster and nimbler software development processes. To name a few of those developed: Adaptive Software Development (ASD), Agile Modeling, Crystal Methods, Dynamic System Development, Lean Development and Scrum. All these methodologies acknowledged that high quality software and more importantly customer satisfaction could only be achieved by bringing "lightness" to their processes. Some of the most used agile methodologies are listed below.

3.1 Extreme Programming (XP)

Extreme programming (XP) has evolved from the problems caused by the long development cycles of traditional development models [8]. The XP process can be characterized by short development cycles, incremental planning, continuous feedback, reliance on communication, and evolutionary design [9]. With all the above qualities, XP programmers respond to changing environment with much more courage. Further according to Williams [10], XP team members spend few minutes on programming, few minutes on project management, few minutes on design, few minutes on feedback, and few minutes on team building many times each day. The term 'extreme' comes from taking these commonsense principles and practices to extreme levels. A summary of XP terms and practices is shown below [8]:

- Planning – The programmer estimates the effort needed for implementation of customer stories and the customer decides the scope and timing of releases based on estimates.
- Small/short releases – An application is developed in a series of small, frequently updated versions. New versions are released anywhere from daily to monthly.
- Metaphor – The system is defined by a set of metaphors between the customer and the programmers which describes how the system works.
- Simple Design – The emphasis is on designing the simplest possible solution that is implemented and unnecessary complexity and extra code are removed immediately.
- Refactoring – It involves restructuring the system by removing duplication, improving communication, simplifying and adding flexibility but without changing the functionality of the program
- Pair programming – All production code are written by two programmers on one computer.
- Collective ownership – No single person owns or is responsible for individual code segments rather anyone can change any part of the code at any time.

- Continuous Integration – A new piece of code is integrated with the current system as soon as it is ready. When integrating, the system is built again and all tests must pass for the changes to be accepted.
- 40-hour week – No one can work two overtime weeks in a row. A maximum of 40-hour working week otherwise it is treated as a problem.
- On-site customer – Customer must be available at all times with the development team.
- Coding Standards – Coding rules exist and are followed by the programmers so as to bring consistence and improve communication among the development team.

The lifecycle of an XP project, shown in Figure 4 [9], is divided into six phases: Exploration, Planning, Iterations to release, Production, Maintenance and Death.
In the *Exploration phase*, the customer writes out the story cards they wish to be included in their program. This leads to *Planning phase* where a priority order is set to each user story and a schedule of the first release is developed. Next in the *Iterations to Release phase*, the development team first iteration is to create a system with the architecture of the whole system then continuously integrating and testing their code. Extra testing and checking of the performance of the system before the system can be released to the customer is done in the *Production phase*. Postponed ideas and suggestions found at this phase are documented for later implementation in the updated releases made at the *Maintenance phase*. Finally the *Death Phase* is near when the customer have no more stories to be implemented and all the necessary documentation of the system is written as no more changes to the architecture, design or code is made.

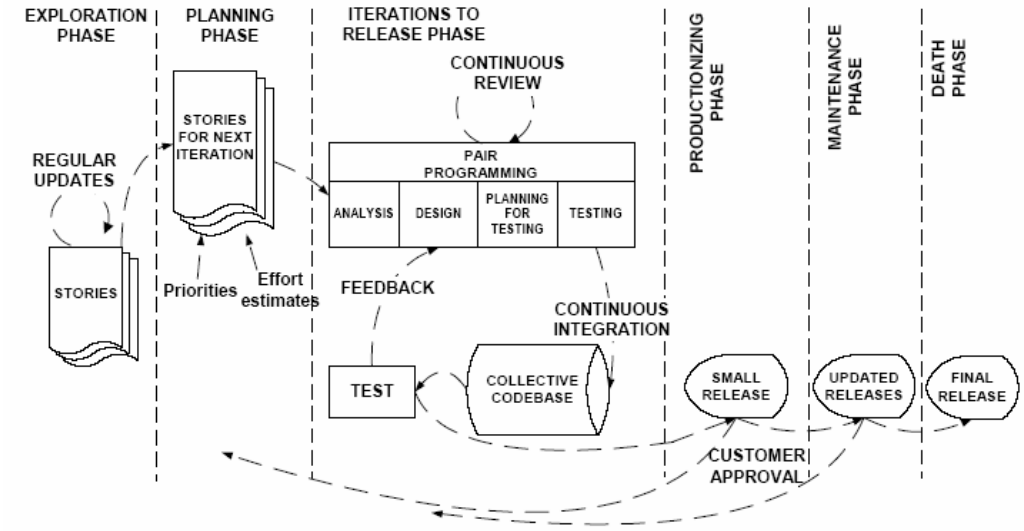


Figure 4: Lifecycle of the XP process [9]

3.2 Scrum

Scrum is an iterative, incremental process for developing any product or managing any work. Scrum concentrates on how the team members should function in order to produce the system flexibility in a constantly changing environment. At the end of every iteration it produces a potential set of functionality. The term 'scrum' originated from a strategy in the game of rugby where it denotes "getting an out-of-play ball back into the game" with teamwork [13].

Scrum does not require or provide any specific software development methods/practices to be used. Instead, it requires certain management practices and tools in different phases of Scrum to avoid the chaos by unpredictability and complexity [12]

Key Scrum practices are discussed below [13] and the Scrum process is shown in Figure 5.

- Product Backlog - This is the prioritized list of all features and changes that have yet to be made to the system desired by multiple actors, such as customers, marketing and sales and project team. The Product Owner is responsible for maintaining the Product Backlog.

- Sprints - Sprints are 30-days in length, it is the procedure of adapting to the changing environmental variables (requirements, time, resources, knowledge, technology etc) and must result in a potentially shippable increment of software. The working tools of the team are Sprint Planning Meetings, Sprint Backlog and Daily Scrum meetings.

- Sprint Planning meeting – Sprint planning meeting is first attended by the customers, users, management, Product owner and Scrum Team where a set of goals and functionality are decided on. Next the Scrum Master and the Scrum Team focus on how the product is implemented during the Sprint.

- Sprint Backlog – It is the list of features that is currently assigned to a particular Sprint. When all the features are completed a new iteration of the system is delivered.

- Daily Scrum – It is a daily meeting for approximately 15 minutes, which are organized to keep track of the progress of the Scrum Team and address any obstacles faced by the team.

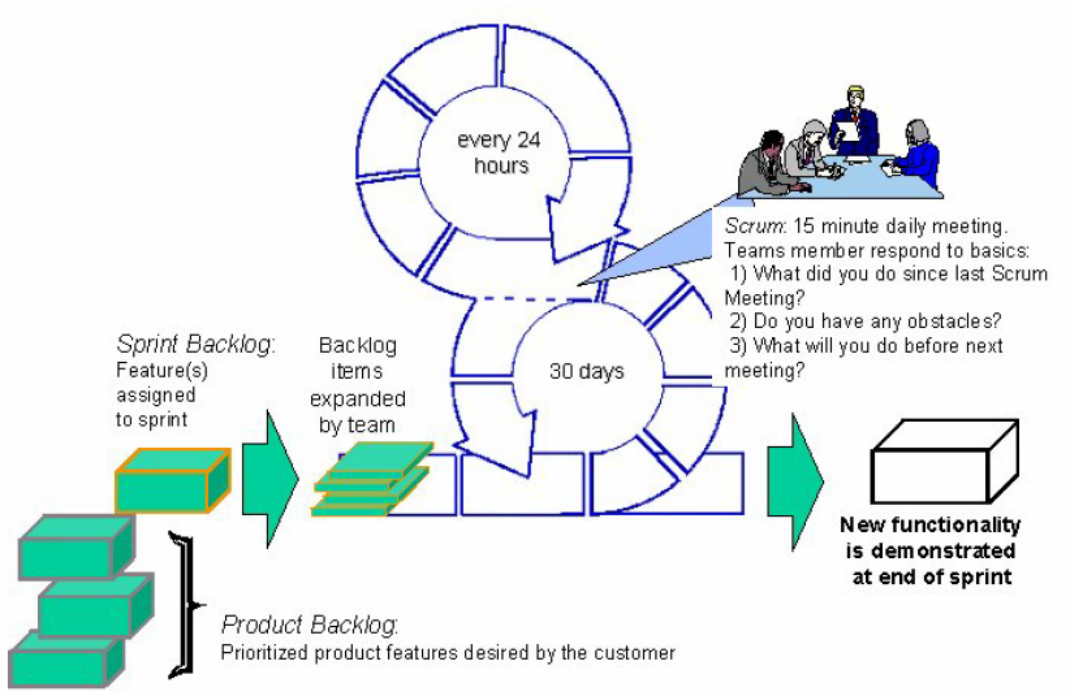


Figure 5: Scrum Process [14]

The Scrum process may change the job description and customs of the Scrum project team considerably. For example, the project manager, i.e. the Scrum Master, does no longer need to organize the team but the team organizes itself and makes decisions on what to do. Ken Schwaber illustrates, "Most management is used to directing the project, telling the team what to do and then ensuring they do it. Scrum relies on self-organization, with the team deciding what to do while management runs interference and removes roadblocks" [11]. Scrum has been successfully used over thousands of projects in 50 organizations producing significant productivity improvement [14]. Rising and Janof [12] suggest that "Clearly, Scrum is not an approach for large, complex team structures, but we found that even small, isolated teams on a large project could make use of some elements of Scrum. This is true process diversity". Recently, efforts have been made to combine XP practices with Scrum project management framework to form an integrated package for software development team [11].However more study is needed to support this package.

3.3 Feature Driven Development (FDD)

Feature Driven Development (FDD) was used for the first time in the development of a large and complex banking application project in the late 90's [15].Unlike the other methodologies, the FDD approach does not cover the entire software development process but rather focuses on the design and building phases [15].

The first three phases are done at the beginning of the project. The last two phases are the iterative part of the process which supports the agile development with quick adaptations to late changes in requirements and business needs. The FDD approach includes frequent and tangible deliverables, along with accurate monitoring of the progress of the report [16].
FDD consists of five sequential steps (Figure 6), an explanation of the different roles and responsibilities used in FDD are explained in Appendix B:

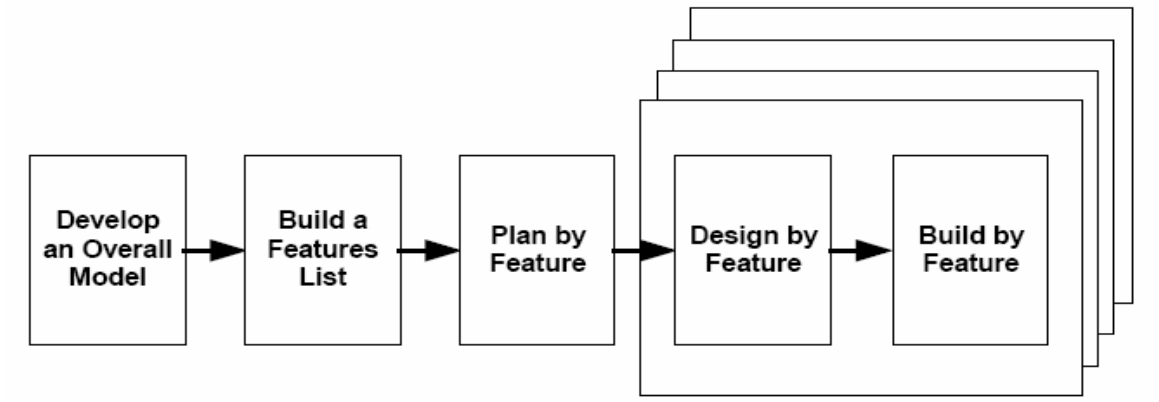


Figure 6: Feature Driven Development processes [15]

- Develop an Overall Model - A high level walkthrough of the system scope and its context is performed by the *domain expert* to the team members and *chief architect*. Documented requirements such as use cases or functional specifications are developed.
- Build a Features List - A categorized list of features to support the requirements is produced
- Plan by Feature - The development team orders the feature sets according to their priority and dependencies and assigned to c*hief programmers*. Furthermore, the classes identified in the first phase are assigned to c*lass owners (individual developers)*. Also schedule and milestones are set for the feature sets.
- Design by Feature & Build by Feature - Features are selected from the feature set and feature teams needed to develop these features are chosen by the class owners. The design by feature and build by feature are iterative procedures during which the team produces the sequence diagrams for the assigned features. These diagrams are passed on to the developers who implement the items necessary to support the design for a particular feature. There can be multiple feature teams concurrently designing and building their own set of features. The code developed is then unit tested and inspected. After a successful iteration, the completed features are promoted to the main build.

3.4 Dynamic System Development Method

The DSDM, Dynamic System Development Method, was developed in the United Kingdom in the mid-1990. It is a blend of, and extension to, rapid application development and Iterative development practices [18]. Martin Fowler, one of the writers of Agile Manifesto, believes, "DSDM is notable for having much of the infrastructure of more mature traditional methodologies, while following the principles of the agile methods approach" [6]. The fundamental idea behind DSDM is to fix time and resources, and then adjust the amount of functionality accordingly rather than fixing the amount of functionality in a product, and then adjusting time and resources to reach that functionality[17]. DSDM consists of five phases (Figure 7):

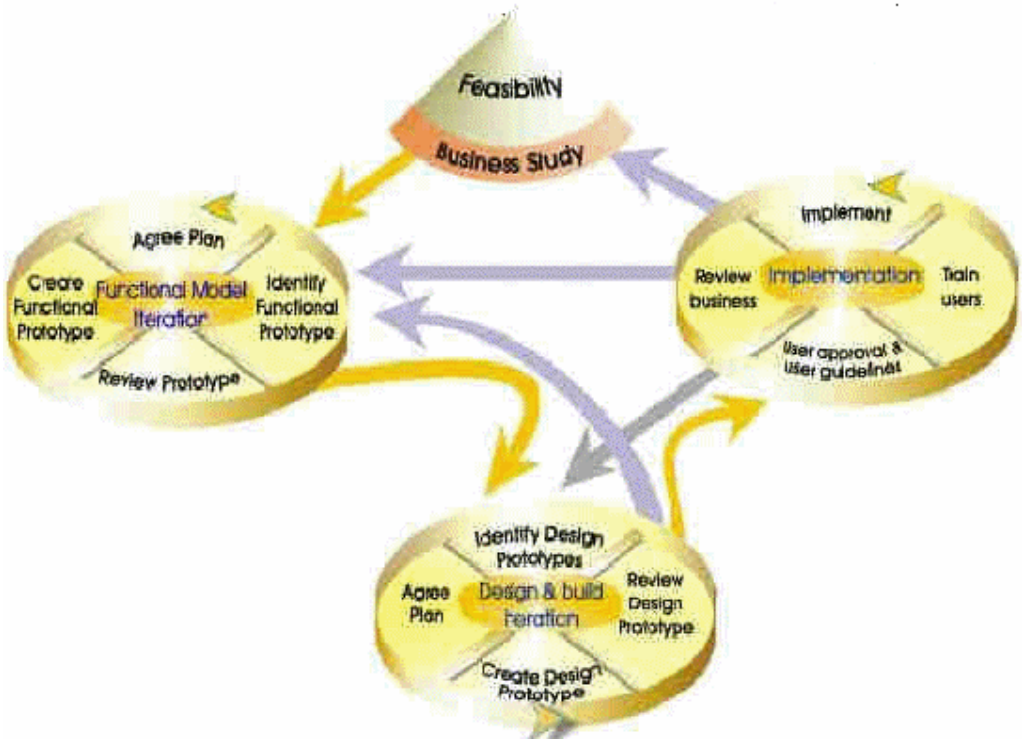


Figure 7: DSDM process diagram [17]

- Feasibility Study – In this phase a decision is made whether to use DSDM or not. This is determined by judging the type of project and, organizational and people issues. In addition, two work products are produced; a feasibility report and an outline plan for development.
- Business Study – The recommended approach to this phase is to organize a workshop to help understand the business domain of the project. The key outputs of this section are System architecture definition and an Outline prototype plan.
- Functional Model Iteration – First iterative phase. This phase involves analysis, coding and prototypes. The results gained from these prototypes are used in

improving the analysis models. The key output is a functional model which consists of the prototype code and analysis models.

- Design and Build Iteration – The system is mainly built in this phase. The design and functional prototypes are reviewed by the users and further development is based on the users' comments.
- Implementation – In this final phase the system is handed over to the users. Training is provided. User Manuals and a Project Review Report. However, the DSDM iterative and incremental nature means that maintenance can be viewed as continuing development. Instead of finishing the project in one cycle, the project can return to any of the phases, Design and Build phase, Functional Model Iteration, or even Feasibility phase so that previous steps can be refined.

There are nine practices that define the ideology and the basis for all activity in DSDM. Some of the underlying principles include active user interaction, frequent deliveries, empowered teams, and testing throughout the cycle. There is an emphasis on high quality and adaptivity towards changing requirements. Like other agile methods, DSDM approaches iterations as short time-boxed cycles of between two and six weeks.

3.5 Adaptive Software Development (ASD)

Adaptive Software Development (ASD), developed by James A. Highsmith, offers an agile and adaptive approach to high-speed and high-change software projects [19]. It is not possible to plan successfully in a fast moving and unpredictable business environment. In ASD, the static plan-design life cycle is replaced by a dynamic speculate-collaborate-learn life cycle.

ASD focal point is on three non-linear and overlapping phases (Figure 8) [18]:

- Speculate - To define the project mission, make clear the realization about what is unclear.
- Collaborate – Highlights the importance of teamwork for developing high-change systems
- Learn – This phase stresses the need to admit and react to mistakes, and that requirements may well change during development.

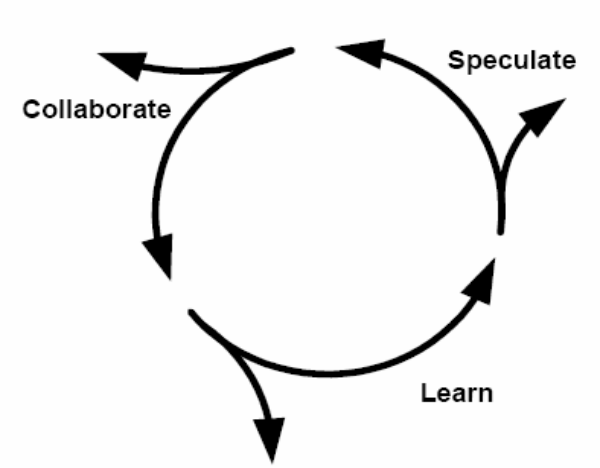


Figure 8: ASD Lifecycle [18]

Since outcomes are naturally unpredictable, Highsmith views planning as a paradox in an adaptive environment. Normally in traditional planning when things do not go to plan it is seen as a mistake that should be corrected. However in an adaptive environment deviations guide us towards the correct solution.

ASD focuses more on results and their quality than the tasks or the process used for producing the results. In an unpredictable environment you need people to collaborate in a certain manner to deal with the uncertainty. Management is more about encouraging communication rather than telling people what to do, so that more creative answers are delivered.

In traditional predictive environments, designs are followed the same way they were laid out, therefore learning is discouraged. Highsmith points out, “In an adaptive environment, learning challenges all stakeholders - developers and their customers - to examine their assumptions and to use the results of each development cycle to adapt the next” [18]. As such learning is a continuous and important feature, one that assumes that plans and designs must change as development proceeds [18].

ASD does not have detailed principles like XP, but rather it provides a framework on how to encourage collaboration and learning within the project. ASD is not presented as a methodology for doing software projects but rather it is an approach or an attitude that must be adopted by an organization when applying agile processes [18].

3.6 Agile Manifesto

In February 2001, seventeen representatives from the different agile methods decided to form an Agile Alliance to better promote their views and what emerged was the Agile 'Software Development' Manifesto. Most of the agile techniques have been used by developers before the alliance but it is not till after the alliance that these techniques were grouped together into a workable framework [20].

The focal values honored by the agilists are presented in the following:

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

The 12 principles of the Agile Software development made by the Agile Manifesto [20]:

- *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
- *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
- *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
- *Business people and developers must work together daily throughout the project.*
- *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
- *Working software is the primary measure of progress.*
- *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
- *Continuous attention to technical excellence and good design enhances agility.*
- *Simplicity--the art of maximizing the amount of work not done--is essential.*
- *The best architectures, requirements, and designs emerge from self-organizing teams.*
- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly*

3.7 General Features and Comparison of Agile Methodologies

Comparison often implies valuing one method over the other. In this section Table 1 below discusses each method using three selected aspects: key points, special features and identified weakness. Key points detail the methods, principles, aspects or solution. Special feature describes one or several aspects of the methods that differentiate them from others. Finally, identified weakness relate to some aspects of a method that have been documented in literature.

Method Name	Key Points	Special features	Identified weakness
ASD	Adaptive culture, collaboration, mission-driven component based iterative development	Organizations are seen as adaptive systems. Creating an emergent order out of a web of interconnected individuals	ASD is more about concepts and culture than the software practice
DSDM	Application of controls to RAD, use of timeboxing and empowered DSDM teams.	First truly agile software development method, use of prototyping, several user roles : “ambassador”, “visionary” and “advisor”	While the method is available, only consortium members have access to white papers dealing with the actual use of the method
XP	Customer driven development, small teams, daily builds	Refactoring - the ongoing redesign of the system to improve its performance and responsiveness too change	While individual practices are suitable for many situations, overall view & management practices are given less attention
SCRUM	Independent, small, self-organizing development teams, 30-day release cycles.	Enforce a paradigm shift from the “defined and repeatable” to the “new product development view of Scrum”	While Scrum details in specific how to manage the 30-day release cycle, the integration and acceptance tests are not detailed
FDD	Five-step process, object-oriented component (i.e. feature) based development.	Method simplicity, design and implement the system by features, object modeling	FDD focuses only on design and implementation. Needs other supporting approaches.

Table 1: General Features of Agile Methods [Table modified from 6, 20]

In the software development viewpoint, ASD is the most abstract method [6]. Its key goal "creating an emergent order out of a web" may be appealing but practitioners may experience difficulties in translating the methods new concept to their use. XP represents practice-oriented viewpoints. It contains a number of empirically validated practices found useful by developers. DSDM is differentiated from the other methods because of its use of prototyping. Also DSDM makes use of user roles such as *ambassador*, *visionary* and *advisor* that other methods do not use. The drawback on using DSDM is that the need to belong to the DSDM consortium in order to gain an access to the white papers discussing different aspects of the method. FDD focuses into a simple five-step approach which consists of identifying, designing and implementing features. FDD assumes that some work has already been done to the project. Finally Scrum is a project management approach that relies on self-organizing independent teams implementing a software project in 30-day cycles called sprints.

One of the main decisive issues in the different agile methods is the size of the development team. XP and Scrum focuses on small teams, preferably less than 10 developers. FDD, ASD and DSDM claim to be capable of up to 100 developers. However, when the development team size gets larger, the amount of documentation is likely to increase, thus making the project "less agile" [21]. When the development group exceeds 20 developers, agilists' put a great deal into solving communication problems. As Alistair Cockburn states, "Good people are key to success with big teams" [22].

3.8 Characteristics of Agile Methodologies

According to Highsmith and Cockburn [24] , "what is new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability. This yields a new combination of values and principles that define an agile world view." Highsmith further transcribes from the book *Agile Competitors and Virtual Organizations* the definition of agility: "Agility... is a comprehensive response to the business challenges of profiting from rapidly changing, continually fragmenting, global markets for high-quality, high-performance, customer-configured goods and services."

The following principles of agile methodologies are seen as the main differences between agile and heavyweight:

People Oriented- Agile methodologies consider people – customers, developers, stakeholders, and end users – as the most important factor of software methodologies. As Jim Highsmith and Alistair Cockburn state, "The most important implication to managers working in the agile manner is that it places more emphasis on people factors in the project: amicability, talent, skill, and communication" [25]. If the people on the project are good enough, they can use almost any process and accomplish their assignment. If they are not good enough, no process will repair their inadequacy [24]. As Highsmith highlights, "… people trump process… " [25].

Adaptive – The participants in an agile process are not afraid of change. Agilists welcome changes at all stages of the project. They view changes to the requirements as good things, because they mean that the team has learned more about what it will take to satisfy the market [6]. Today the challenge is not stopping change but rather determining how to better handle changes that occur throughout a project. “External Environment changes cause critical variations. Because we cannot eliminate these changes, driving down the cost of responding to them is the only viable strategy” [24].

Conformance to Actual – Agile methodologies value conformance to the actual results as opposed to conformance to the detailed plan. Highsmith states, “Agile projects are not controlled by conformance to plan but by conformance to the business value” [26]. Each iteration or development cycle adds business value to the ongoing product. For agilists, the decision on whether business value has been added or not is not given by the developers but instead by end users and customers.

Balancing Flexibility and Planning – Plans are important, but the problem is that software projects can not be accurately predicted far into the future, because there are so many variables to take into account. A better planning strategy is to make detailed plans for the next few weeks, very rough plans for the next few months, and extremely crude plans beyond that [19]. In this view one of the main sources of complexity is the irreversibility of decisions. If you can easily change your decisions, this means it’s less important to get them right – which makes your life much simpler. The consequence for agile design is that designers need to think about how they can avoid irreversibility in their decisions. Rather than trying to get the right decision now, look for a way to either put off the decision until later or make the decision in such a way that you will be able to reverse it later on without too much difficulty [27].

Empirical Process – Agile methods develop software as an empirical (or nonlinear) process. In engineering, processes are either defined or empirical. In other words, defined process is one that can be started and allowed to run to completion producing the same results every time. In software development it can not be considered a defined process because too much change occurs during the time that the team is developing the product. Laurie Williams states, “It is highly unlikely that any set of predefined steps will lead to a desirable, predictable outcome because requirements change technology changes, people are added and taken off the team, and so on” [28].

Decentralized Approach – Integrating a decentralized management style can severely impact a software project because it could save a lot of time than an autocratic management process. Agile software development spreads out the decision making to the developers. This does not mean that the developers take on the role of management. Management is still needed to remove roadblocks standing in the way of progress. However management recognizes the expertise of the technical team to make technical decisions without their permission.

Simplicity – Agile teams always take the simplest path that is consistent with their goals. Fowler states, “They (agile teams) don’t anticipate tomorrow’s problems and try to defend against them today” [6]. The reason for simplicity is so that it will be easy to change the design if needed on a later date. Never produce more than what is necessary and never produce documents attempting to predict the future as documents will become outdated. “The larger the amount of documentation becomes, the more effort is needed to find the required information, and the more effort is needed to keep the information up to date” [23].

Collaboration – Agile methods involve customer feedback on a regular and frequent basis. The customer of the software works closely with the development team, providing frequent feedback on their efforts. As well, constant collaboration between agile team members is essential. Due to the decentralized approach of the agile methods, collaboration encourages discussion. As Martin Fowler describes, “Agile teams cannot exist with occasional communication. They need continuous access to business expertise” [6].

Small Self-organizing teams – An agile team is a self organizing team. Responsibilities are communicated to the team as a whole, and the team determines the best way to fulfill them. Agile teams discuss and communicate together on all aspects of the project. That is why agility works well in small teams. As Alistair Cockburn and Jim Highsmith highlight, “Agile development is more difficult with larger teams. The average project has only nine people, within the reach of most basic agile processes. Nevertheless, it is interesting to occasionally find successful agile projects with 120 or even 250 people” [25].

4.0 Limitations of Heavyweight Methodologies

The main difference between heavyweight and agile methodologies is the acceptance of change. It is the ability to respond to change that often determines the success or failure of a software project [28]. Heavyweight methods freeze product functionality and disallow change. However one of the key, philosophical constructs making agile processes successful in today's market is its response to change at any stage of the project. It makes it very difficult to implement a predictive process or to provide a set of stable requirements in this volatile and constantly changing environment. Michael Dell contributes to this by stating, "..the only constant is change" [22]. Martin Fowler and Jim Highsmith founders of the agile manifesto mention that, "Facilitating change is more effective than attempting to prevent it. Learn to trust in your ability to respond to unpredictable events; it's more important than trusting in your ability to plan for disaster," [30]. Furthermore, Boehm [30] and Jones [32] both concluded that during their project development experience, requirements change at 25% or more.

A research study was conducted by a Standish Group of 365 respondents and regarding 8,380 projects representing companies across major industry segments. From their findings, 16.2% of the projects were completed on-time and on-budget with all features and functions specified. However 52.7% of the projects are completed but over-budget, over the time estimate and offering less features and functions while 31.1% of projects were canceled at some point during the development cycle [31] (see Figure 9). The study further reveals that the three major reasons that a project will succeed are user involvement, executive management support, and a clear statement of requirements.

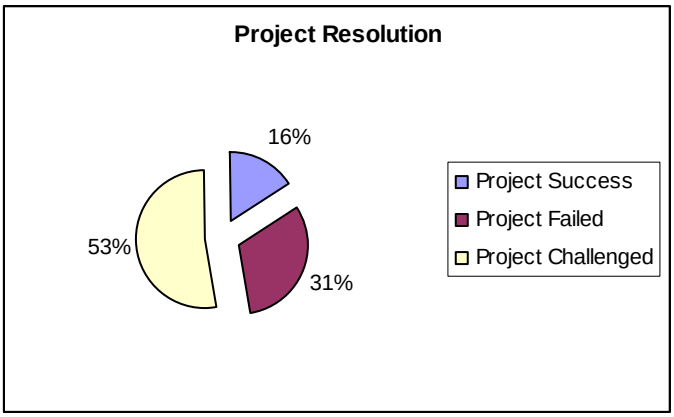


Figure 9: Project Resolution [31]

Another limitation of heavyweight methodologies is the handling of complexity. "Complex rules and regulation give rise to simple stupid behavior," says the former CEO of Visa International [6]. The approach to plan everything and then to follow the plan works smoothly for stable and less complex environment but for larger and more complex environments, this technique would fall apart.

The solution to this problem lies in simplicity. As Dee Hock rephrases his statement to, "Simple, clear purpose and principles give rise to complex, intelligent behavior," [20]. Some companies are using simple rules to survive complex and turbulent markets. For example, Jack Welch CEO of General Electric (GE) transformed his company from a market value of US$12 billion to one of the world's most valuable companies with a value of nearly US$500 billion. His strategy for success according to Stanford University professor, Kathleen Eisenhardt, was based on 2 simple rules [33]. First, Welch developed a simple method for analyzing his company's strengths and weaknesses. He then sold or eliminated businesses that were not promising enough to earn a No. 1 or No. 2 spot in each industry. Next rule, Welch administered the reduction of unnecessary tasks and introduced a process called Work-Out, during which employees of all levels would meet to focus on a problem or an opportunity. If any valuable ideas were introduced during this meeting then regardless of its source they would act on it rapidly [33]. Agilists promote the same idea. As Fowler and Highsmith mention "In an agile project, it's particularly important to use simple approaches, because they are easier to change. It's easier to add something to a process that's too simple than it is to take something away from a process that's too complicated," [20].

Another finding by the Standish group shows that 45% of the features present in an application are never used (see Figure 10). This is another reason for making the design and code as simple as possible as this explains how nearly half of the software production and added complexity were all not needed.

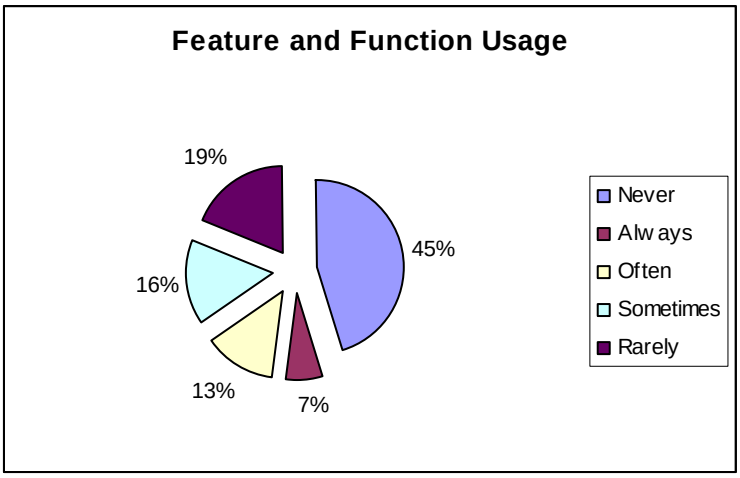


Figure 10: Feature and Function Usage [31]

As mentioned earlier, Kathleed Eisenhardt suggests that instead of following complex processes, using simple rules to communicate strategy is the best way to empower people to seize fleeting opportunities in rapidly changing markets. With simple rules, work teams were able to continuously improve the processes and products without detailed guidance or complex processes [34]. Lean Manufacturing and Total Quality Management (TQM) have a set of rules that have been tested and proven over the last two decades to be useful to software development [35]. Lean Software Development is not a

management or development methodology per se, but it offers principles that are applicable in any environment to improve software development.

The application of the rules may have changed slightly from different industries but the underlying principles are still the same. Mary Poppendieck shows how agile methodologies follow the same set of rules as the Lean manufacturing and TQM unlike heavyweight methodologies. She goes on to explain how the basic principles of Lean Manufacturing and TQM are tantamount to the basic principles of Agile Methodologies. Explanations of some of the rules and their similarities to agile are explained below, for a list of all the rules of Lean Manufacturing and TQM please refer to Appendix C.

The first rule of Lean Manufacturing and TQM is elimination of waste. That is, eliminate anything which does not add value to the final product. Documents, diagrams and models produced as part of the software development must be minimized because once a working system is delivered the user may care little about these deliverables. Agile methodologies follow the same rule for their processes.

The second rule of Lean Manufacturing and TQM is that inventory is waste. Inventory consumes resources, slows down response time and becomes obsolete. The inventory of software development is documentation, excess documentation creates a waste of time in producing and reviewing the documents. Rather than having a 100 page detailed specification, write a 10 page set of rules and guidelines. This is what agile methodologies rigorously maintain, documentation should be kept to minimal.

The third rule of Lean Manufacturing and TQM is to maximize flow. Rather than taking months to show the customer the final product, use an Iterative development where small but complete portions of a system are designed and delivered throughout the development cycle. Similar to agile methods this technique allows the customer to have a better idea of how the software works.

The fourth rule of Lean Manufacturing and TQM is pull from demand and deciding as late as possible. Software development practices which keep requirements flexible and as close to system delivery as possible can provide a significant competitive advantage in a changing environment. Similarly, agile methodologies are designed to respond to change, not predict it, and have the ability to make decisions as late as possible.

The fifth rule of Lean Manufacturing and TQM is to empower workers, to provide both the tools and the authority for people other than mangers to make decisions. This is one of the problems with heavyweight documentation is that it attempts to make all of the decisions for developers. However agile methodologies give developers guidance as well as freedom to make the detailed design and programming decisions. Mary mentions, “It is always better to tell developers what needs to be done, not how to do it” [35].

Sequential versus Iterative development is another reason traditionalist and agilists are very different. Heavyweight methodologies put customer feedback and testing at the last stage of their project lifecycle. Agilists believe otherwise, that they should be embedded

as a daily exercise. The key to Iterative development according to Fowler is to frequently produce working versions of the final system that have a subset of the required features. These working system are short on functionality but should be faithful to the demands of the final system [6]. Heavyweight methodologies produce documents to show the users the requirements of the software. This sort of method could hide all sorts of flows. Agilists believe that there is nothing like a tested, integrated system for bringing a forceful dose of reality into any project. “When people actually sit in front of a system and work with it, then flaws become truly apparent: both in terms of bugs and in terms of misunderstood requirements” [6].

The Standish Group International found in their study of 23,000 projects that the delivery of software components early and often with short time frames, increase the success rate. “Growing (instead of “developing”) software engages the user earlier and confers ownership” [31]. Another study by Alan MacCormack, a Harvard Business School Professor, on the development of software process on 30 projects showed that an early release of the evolving product design to the customer is one of the factors to a successful project. MacCormack states, “The most striking result to emerge from the research conducted concerned the importance of getting a low-functionality version of the product into the customers’ hands at the earliest opportunity,” [36]. Both studies by MacCormack and Standish group heavily prove that the foundation processes of agility; short iterative development, continuous rapid feedback and testing, and incremental development dramatically improve the quality of the software.

Another important criticism against heavyweight methodologies is their treatment of people involved in developing a process. Traditional methodologies treat people as predictable components similar to what they treat their processes. In Alistair Cockburn paper, *Agile Software Development: The People Factor*, he concludes from his studies of software projects that people are the most important factor in software development [25]. Alistair Cockburn is the most explicit in his people-centric view of software development, but the problem is that methodology has been opposed to the notion of people as the first-order in project success [6]. However this creates a strong feedback effect that if you treat all the developers as plug compatible programming units and not as individuals this lowers the morale and creativity of the developers. As well the Standish Group outlines in their report a recipe for success from their research. The first three most important factors for a successful project are executive support, user-involvement, and experienced project management [17].

Agile methodologies focus on the talents and skills of individuals and molds processes to specific people and teams, not like heavyweight methods where all tasks and roles are assigned to individuals and it is expected that the individuals will perform their tasks accordingly. To strengthen this argument Marcus Buckingham and Curt Coffman interviewed 80,000 managers in 400 companies over a 25 year period for a research program by the Gallup organization. In their book, they mention that it is not the organizations that employ traditional processes value people less than agile ones, it is that they view people and how to improve their performance differently. They state, “Rigorous processes are designed to standardize people to the organization, while agile

processes are designed to capitalize on each individual and each team's unique strengths,"[37].

It is a big belief among agile process proponents that people can respond quicker and transfer ideas more rapidly when talking face-to-face than they can in heavyweight methodologies when reading or writing documentation. When developers talk with customers and sponsors, they could work out difficulties, adjust priorities, and examine alternate paths forward in ways not possible when they are not working together. According to Cockburn the most significant single factor is "communication". He illustrates in the figure below that the communication effectiveness drops as modalities and timing are removed [38]. Hewlett-Packard and IBM were early to observe the effectiveness of informal meeting places, but now its part of the industry to have an effective design environment actively encourage and permit ad hoc meetings of small groups [38].

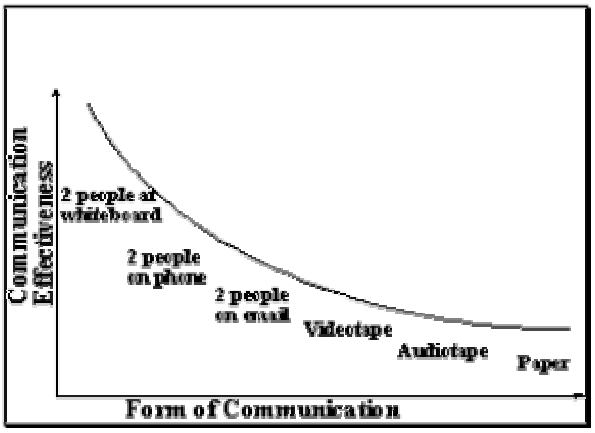


Figure 11: Modes of Communication [53]

Another argument between agile and heavyweight methodologies is the measurement of project success. A predictive heavyweight project considers handing a project that is on-time and on-cost to be a success [17].However agilists measures project success by questioning if the customer got software that is more valuable to them than the cost put into it. According to Martin Fowler, "A good predictive project will go according to plan, a good agile project will build something different and better than the original plan foresaw" [6].

5.0 Limitations of Agile Methodologies

What are the risks and shortcomings of the agile methods? Techniques of agile methods have been around since 30 years ago. Larry Constantine states, “IBM touted so-called Chief programmer teams small, agile groups headed by a strong technical leader who could hold the implicit architecture in mind without resorting to much explicit design” [39]. The objective of IBM was to be able to have working code at all times and gradually grow it to become a working system, this is similar to the objective of agile methodologies. However this chief programmer technique enjoyed early victories for a while but later faded away. The reason for this as Constantine mentions, “not every problem can be sliced and diced into the right pieces for speedy incremental refinement.” [39]. So would agile methodology work this time?

The biggest limitation of agile methodologies is how they handle larger teams. Cockburn and Highsmith both conclude that “Agile development is more difficult for larger teams…as size grows coordinating interfaces become a dominant issue,” [25]. Both Larry Constantine and Martin Fowler also believe that agile with face-to-face communication breaks down and becomes more difficult and complex with developers more than 20 [39,6]. In contrast, heavyweight and plan-driven methods scale better to large projects.

Barry Boehm disagrees to a degree with the first principle of agile manifesto which states, “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software” [20]. He states that, “Overfocus on early results in large systems can lead to a major rework when the architecture doesn’t scale up” [37]. Boehm also contends that a plan-driven process is most needed for high assurance software. Even the originator of agile modeling Scott Ambler mentions, “I would be leery of applying agile modeling to life-critical systems” [41]. Heavyweight traditional goals such as predictability, repeatability, and optimization are often characteristics of reliable safety critical software development. Most agile techniques do not support traditional walkthroughs and code inspections during the life-cycle, it emphasizes on pair programming and informal reviews as their quality control mechanism. This kind of technique has not yet been proved adequate enough for strict regulations of critical software. Alistair Cockburn questioned this by stating, “How agile can we be, given that this is going to be critical, reliable and safe?” [21]. Moreover, Martin Fowler mentions that agile methods provide workable solutions only for “business software” [6].

Alistair Cockburn and Jim Highsmith emphasize severe critical people factors, such as amicability, talent, skill, and communication are the most important factors to a success of a project [25]. Boehm contends noting that, “A significant consideration here is the unavoidable statistic that 49.9999 percent of the world’s software developers are below average,” [40]. As well Larry Constantine’s contributes to this problem for agile methods by stating, “There are only so many Kent Becks in the world to lead the team. All of the agile methods put a premium on having premium people,” [42]. While agile does not

require uniformly high-capability people, it relies on tacit knowledge embodied by the team, rather than writing the knowledge down as documentation. Boehm points out that there is a risk that this may lead to architectural mistakes that cannot be easily detected by external reviewers due to the lack of documentation. However plan-driven or heavyweight methods reduce this risk by investing in life-cycle architectures and plans and using theses to facilitate external expert reviews even though these plans may be obsolete or expensive to change if a change occurs [42].

This debate about whether or not agile methodologies require having creative and skillful people to be effective, leads to another argument. Using "premium" people could make just about anything to happen and that specific type of methodology is not important when you work with "premium" people. This suggests that perhaps the success of agile methods could be attributed to the team of good people, rather than the practices and principles. Alistair Cockburn agrees with this argument but explains, "If the people on the project are good enough, they can use almost any process and accomplish their assignment. If they are not good enough, no process will repair their inadequacy – "people trump process" is one way to say this," [25]. Agilists treat people as first-order project success; they have strong belief of people-oriented approach in contrast to process-oriented approach.

Agile methodologies have a strong emphasis on customer involvement. The customer is considered as part of the development team throughout the whole development of the software. The Standish Group research topped this by providing the second most important factor for a project success is user involvement according to IT executive mangers opinion. According to Boehm, "Agile methods work best when such customers operate in dedicated mode with the development team, and when their tacit knowledge is sufficient for the full span of the application" [40]. Again these methods risk tacit knowledge shortfall. If you have one customer participant then unless they are committed, knowledgeable, collaborative and empowered then there is some chance that you would have a unified set of requirements. However if you have many customers you would have different viewpoints and conflicts between them. This risk could be reduced in plan-driven methods by using documentation, planning, architecture reviews and independent expert project reviews to compensate for on-site customer negligence [40].

Another factor of agile methodologies that could cause problems is the interpretation of the agile manifesto principle, "Working Software over Comprehensive Documentation" [20]. Boehm questions the applicability of agiles' emphasis on simplicity. Based on XP's concept of YAGNI precept: "You Aren't Going to Need It," believes that doing extra work to get rid of architectural features that do not support the current version. This might cause misconceptions for the developers. This approach can be useful when the future requirements are highly unpredictable. However where future requirements are predictable, Boehm states "this practice not only throws away valuable architecture support for them, it also creates problems with the customers who want developers to believe that their priorities and evolution requirements are worth accommodating" [40].

Product and project documentation is a topic that has drawn a lot of attention to agile methods. Is any documentation needed at all and if so how much is enough? Scott Ambler points out, “Organizations demand more documentation than needed, and that documentation is a poor form of communication” [21]. He also commented that documentation becomes out of date and should be updated only “when it hurts”. However documentation is needed in order to retain critical information over time. Barry Boehm mentions, “A documented project makes it easier for an outside expert to diagnose problems” [21]. Proposing no documentation increases a risk when considering maintenance and usage aspects, agilists base an assumption that teams will stay together until the very end of the software development which is not likely to happen in most cases.

According to informants in the agile process community, agile methods seem to be light on the user side of software i.e. User interface design and usability. As one of these informants Alistair Cockburn mentions, “It is not a weak point- it is an absence” [42]. When it comes to user interface design, agile processes prefer simplistic forms or iterative paper prototyping rather than model-driven design. Agilists believe that testing of this user interface is labor intensive and time consuming. However Larry Constantine states that, “User or client reactions to paper prototypes are no substitute and can even be completely misleading-what people will say they like or claim is feasible when they see a paper will often prove unworkable in practice”[42]. In short Larry Constantine adds that user centered design qualifies as an agile process as it supplies an effective and efficient scheme for designing highly useable user interface.

6.0 Implementation of Agile Methodologies

In software development there exists a tension between quality, cost and time. Barry Boehm states that, “As we progress from analysis, through to design, coding, testing and production, the cost of fixing a problem increases exponentially” [42]. The greatest increase in cost is when fixing the problem after product introduction, a cost of approximately 60 to 100 times more than eliminating the problem in the design phase. Boehm suggests to reduce these costs, use heavyweight methodologies so that more time is spent in upfront requirements gathering.

Alistair Cockburn disagrees with Boehm’s statement and reports, “As time goes by and the program gets bigger, it costs LESS to implement a change with XP than with your traditional methodology” [43]. In addition, Kent Beck argues that the “cost of change” curve is said to be flat in agile modeling [42]. Moreover to strengthen this conviction they show several XP practices to ensure that the cost associated with this curve is kept to minimal [42]:

- Unit Testing and Test-Driven Development ensures that bugs and errors are found quickly and early so that it would be cheaper to fix.
- On-site customer and functional testing ensure the analysis and specification of the system is up-to-date and precise with business requirements.
- Pair programming allows two developers working together on one computer, which increases the chances of finding bugs and leads to a simpler design
- Refactoring and “once and only once” increases design consistency and adds more simplicity and flexibility to the structure. This ensures that the system is well-designed and easy to change.
- Regular releases gives the customer feedback and forces the team to make the “release to production” and maintenance phases as cheap as possible.

The above agile principles attack the roots of the high cost of fixing errors (with good specifications, good designs, good implementation and fast feedback). But according to Laurie Williams this does not mean that agile processes decrease or increase the cost of developing compared to heavyweight [40]. In Figure 12 below, Williams shows two theoretical graphs to illustrate this. Figure 12 graph *a*, represents the expense of traditional methods over time and mentions that most of the expense is spent on new development and little expense on revision which is done during the development cycle. Conversely, Figure 12 graph *b* represents an agile (XP) method project’s expense. Here the opposite occurs, demonstrating more spending on the revision and less on the development. According to these results both graphs indicate the same level of expense over similar time periods. William states, “Strong anecdotal evidence suggests that the additional revision does not exceed the expense that would have been incurred had extensive up-front requirements engineering, planning and designing” [40].

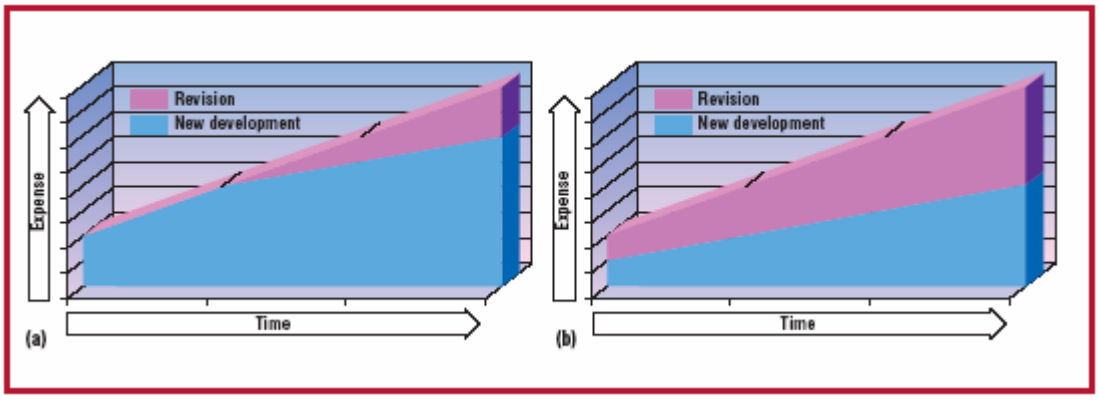


Figure 12: Cumulative Expense for Heavy and Agile Development [40]

Shine Technologies, Victoria, Australia conducted a global survey of experiences using agile methodologies of diverse organizations from Online Computer Library to NASA. From the survey results, 95% of the respondents believed that costs were the same or less when using agile methods compared to when they used a traditional methodology. This goes on to support Laurie Williams theoretical view that expense is the same for heavyweight and agile methods over similar time. However the respondents also found stunning improvements in productivity, quality and business satisfaction. Some of the highlights of the findings are below [44]:

- *84.7% of respondents rated their Agile knowledge as average or above. We have classified these respondents as "knowledgeable" for the purpose of the survey*
- *49%* stated that costs were reduced or significantly reduced*
- *46%* stated that costs were unchanged, resulting in 95% stating that there was either no effect or a cost reduction*
- *Only 5%* stated that Agile processes had a negative effect on cost*
- *93%* stated that productivity was better or significantly better*
- *88%* stated that quality was better or significantly better*
- *83%* stated that business satisfaction was better or significantly better*
- *Knowledgeable respondents were vastly more in favor of Agile processes. Only 1.8% of knowledgeable respondents found productivity degradations, but this increased to 3.1% when taken across all respondents.*

**Of knowledgeable respondents as identified in Question 1 of the survey*

6.1 Implementing Agile Processes in Software Organizations

Software has been part of modern society for more than 50 years, likewise so have software development processes [6]. However agile methods oldest methodology was SCRUM and DSDM and they were not defined till mid-1990. Even though each methodology has excellent anecdotal evidence and research results that their method works, not enough statistical and metric proof has been gathered [45]. Geoffrey Moore, in

his book *Crossing the Chasm*, describes five types of profiles of technology adopters: Innovators who pursue new concepts aggressively; early adopters who pursue new concepts very early in the lifecycle; early majority wait and see before buying into a new concept; the late majority who are concerned about their ability to handle a new concept; and laggards who do not want anything to do with new approaches [47]. According to Scott Ambler, people that fit the innovator or early adopter would adopt agile techniques. Moreover, since there is sufficient anecdotal evidence, the early majority are starting to adopt agility to their organization. Furthermore he adds, "It will take several years, perhaps even a decade, until we have incontrovertible proof that agile software development work in practice" [45].

DaimlerChrysler was the first organization to use agile methods that introduced XP practices with the Chrysler Comprehensive Compensation (C3) project, which is a very successful payroll system implemented in Smalltalk. The C3 project began in January 1995 under a fixed priced contract and a year later failed to deliver a proper working payroll system [46]. Kent Beck, the developer of XP, was called in to help with performance tuning of C3 project and found that the code was poorly factored, there was no repeatable tests, and the management had lost confidence in the project. Beck threw away all the previous code and the fixed-price contract was cancelled. He reorganized the team and made up the rules of XP that they had to follow: "putting customer on-site to work with the developers, sharing code techniques, pairing developers, performing automated unit testing and editing code frequently to keep it simple"[48]. All these modifications enhanced and developed a successful payroll system that did more than what was needed. Chrysler still uses the XP concept as Christen Wege, portal and Web application architect, mentions, "Today, Stuttgart, Germany-based DaimlerChrysler AG still uses extreme programming within several application development groups in the U.S. and Germany" [46].

One of the most difficult tasks involved with using agile processes is successfully introducing them into an organization that has been using their traditional organization structure for years. "Part of [the Big Design Up-Front] culture is the creation of fiefdoms within the program organization. Adopting [agile processes] will radically change the functions of the organization within the program and consequently change the staff and funding profiles of the organizations" [6]. Some of the traditional roles such as the Quality Assurance and testing would resist the change as more attention and work is needed from these roles after each iteration in an agile process. Management are uncomfortable with not having documents to judge the progress of their project and not having a final commitment date of delivery with a bottom line cost [6]. Still accordingly to Chris Dial, an analyst at Forrester Research Inc, "organizations are increasingly turning to new techniques to make the most of the smaller development teams and contend with more complex, distributed applications" [46].

A Singapore lending project was declared undoable until Jeff De Luca, a project director of Nebulon, a leading information technology firm in Melbourne, took on the project using the agile methodology Feature-Driven Development (FDD). Previously the

deliverables included 3,500 pages of use cases, an object model with hundreds of classes, thousands of attributes (but no methods) and no code. De Luca used techniques such as keeping code simple, testing often and delivering small features of the application as they are ready. Within 2 months De Luca's team was producing demonstrable features for the client and 4 months later the project was completed and under budget. When asked what his key to success De Luca responded was, "The key is having good people, good domain experts, good developers and good chief programmers. No process makes up for a lack of talent and skill" [49]. This example shows a clear example of why working code is the ultimate arbiter of real progress. As Jim Highsmith states, "In the end, thousands of use cases and hundreds of object model elements did not prove real progress" [49].

Caterpillar Financial Services Corp. also used an agile technique to develop a critical web-based financial system for its dealers all over the world. The success of this project according to Tom DePauw, manager of IT at Caterpillar, was using agile methods to build small, usable parts of Java based applications early, rather than one large application at the end of the project [50]. Furthermore a large US based financial institution agrees that the need to produce functional parts of the application regularly to the customer will drive your company to consider agile methodologies. They state "Customers want applications in 90 days now, no matter how complex they are, and you can't do that with traditional methods" [51].

However, there are some downfalls in using agile methodologies in the software industry and one of them is their over emphasis towards customer collaboration. According to Erkki Vuorenmaa, manager of IT company in Finland, getting business people involved in the development process is very "irritating" and awkward job, and without the determined "good" customer it would be hard to develop a quality software [48]. Another criticism of agile methods is concerning project costs. Agile projects have no fixed price or fixed schedule and projects are open-ended and evolve as requirements change. Therefore it becomes harder for the manager and customer to accept this technique as customers would rather know the total cost of the project and overall project schedule beforehand. On the other hand Alistair Cockburn pointed out that agile and fixed price are not mutually exclusive. He came up with the version of agility through a succession of fixed price projects. Cockburn explains, "In fixed price projects the price is usually fixed to low, so you want to do everything you can to boost productivity, and that includes using an agile process" [43].

Motorola's experience with agile methods in its development organization found that it was not useful for global development projects. Senior architect of Motorola believed that the agile method [Extreme Programming] values small teams and that was not always possible. Surprisingly some believe that after mangers hear the name 'extreme programming' they get turned off. However, on the upside, agile methods provide short daily meetings that would lead to better continual feedback; this keeps the cost to minimal. As a manager at Sunoco Inc says, "If the consultant is incompetent or the technology is wrong, I get the first indication after 30 days. I'm cutting my losses quickly" [51].

Agile practices have been widely accepted in many organizations due to their similarities to CMM (Capability Maturity Model) standards. The development of the CMM has become a standard to well-defined and well-documented software development processes for organizations to follow to succeed in their project. Laurie Williams adds, “Many CMM or ISO 9000 now think that partial adoption of agile practices, when handled with care, might increase their efficiencies without damaging their certifications” [28]. Mark Paulk, from the Software Engineering Institute, states, “XP has good engineering practices that can work well with the CMM and other highly structured models. The key is to carefully consider XP practices and implement them in the right environment” [53]. He goes on to show that certain agile practices of XP are similar to Level 2, 3 and some of 4 practices of CMM (for the complete table of CMM standards refer to Appendix D). For example, XP meets CMM Level 2 *requirements management* condition through its use of stories, an onsite customer, and continuous integration. XP address *software project planning* in the planning game and small releases. XP’s practices with “big visual chart”, project velocity, and commitments for small releases meet *Software project tracking and oversight* in CMM level 2. In CMM level 3 several XP practices address *software product engineering* such as metaphor, simple design, refactoring, coding standards and unit testing. XP’s strong emphasis on communication and pair programming consecutively addresses *intergroup coordination* and *peer reviews* of CMM level 3. Beyond level 3, XP only address as few of the Level 4 and 5 key process areas [53]. Moreover this popularity of Extreme programming to the level of alchemy was supported by respected people like Tom DeMarco that once stated that, “An organization employing Extreme Programming moved from CMM Level 1 to CMM Level 4 within 5 months” [53].

6.2 Agile Methods and Offshore Development

In the past few years, many companies have turned to offshore software development for faster, better, and cheaper development teams. According to 2003 CIO Magazine survey, lower cost was cited by 78% of the IT executives as “... the main reason for outsourcing offshore. The great savings were realized in the areas of labor costs (86%) and reduced project timelines/time-to-complete (37%). Other benefits experienced as a result of offshore outsourcing included increased IT department productivity (44%), competitive advantage (30%), and internal customer satisfaction (20%)” [54]. With offshore development comes no notion of physical proximity, and most offshore development favor the plan-driven approach where business analysis, detailed requirements and design are done at the front office(on-shore) and sent to the back office to be constructed. This arrangement comes with a challenge for agile methodologies. Due to time zone difference and separated by thousand of miles decreases the volume of communication between offshore and onshore teams. However agile software methodologies place strong emphasize on the importance of communication and improving of communication between people involved in software development. This leads to the question: Are these two compatible with one another, or are organizations going to have to choose between “going agile” and “going offshore”?

Offshore development is created with challenges and using parts of agile development appear to make offshore even harder to manage. For the past few years, ThoughtWorks has operated a lab in Bangalore India to support software development projects in North America and Europe. Martin Fowler provides some insights on his experience and lessons learned in doing offshore agile development rather than the traditional plan-driven methodologies whilst working with ThoughtWorks [56].

- *Use Distributed Continuous Integration to Avoid Integration Headaches* - If practiced with discipline, the process by which developers integrate their code and build the entire system whenever they have made changes and fix errors before they become hard to find, should reduce or eliminate configuration management issues.
- *Have Each Site Send Ambassadors to the Other Sites* – As mentioned before agile methods strongly rely on face to face human interaction. A solution to this is to bring onshore team members to the offshore site. ThoughtWorks ensured that at all times, there was someone from the US team present in India to facilitate the communication. The benefits of having an ambassador are to help everyone communicate to the right people and provide a business context to the offshore team.
- *Use Contact Visits to build trust* – Ambassadors are semi-permanent people, but this is not enough. There should be more visits by the offshore team to visit the onshore team. These visits help to create and maintain relationships which need to be in place for remote communication to work effectively.
- *Don't Underestimate the Culture Change* – According to Fowler, one of the hardest parts of introducing agile methods into organizations is the culture change it causes [56]. The main reason why companies don't adopt agile methods offshore is because of the command and control model in which many Asian companies follow.
- *Use Test Scripts to Help Understand the Requirements* – Acceptance tests help to communicate and clarify the requirements between offshore and onshore team members. Writing out the tests forces the offshore development team a concrete target to aim at.
- *Use Regular Builds to Get Feedback on Functionality* - Fowler suggests that the quicker the customer can look at a partial functional, the quicker they can spot any miscommunications.

Other methods that Fowler described to get a smooth agile offshore development - Use regular short status meetings; Use short iterations; Use an iteration planning meeting that tailored for remote sites; When moving a code base, bug fixing makes a good start; Separate teams by functionality bit by activity; and Expect to need more documents.

There are still many differences in opinion about the cost and benefits of using offshore development. Offshore developments weakness is culture and distance from the business. However agile methodologies work best with close communication and an open culture. Its too hard to prove one approach better than the other. What is seen is growing qualitative feedback on the benefits of agility and offshore development.

7.0 Comparison of Agile and Heavyweight

Traditional development approaches have been around for a very long time. Since its introduction the waterfall model (Royce 1970) has been widely used in both large and small software projects and has been reported to be successful to many projects. Despite the success it has a lot of drawbacks, like linearity, inflexibility in changing requirements, and high formal processes irrespective of the size of the project. Kent Beck took these drawbacks into account and introduced Extreme Programming, the first agile methodology produced. Agile methods deal with unstable and volatile requirements by using a number of techniques, focusing on collaboration between developers and customers and support early product delivery. A summary of the difference of agile and heavyweight methodologies is shown in the table below.

	Agile Methods	Heavy Methods
Approach	Adaptive	Predictive
Success Measurement	Business Value	Conformation to plan
Project size	Small	Large
Management Style	Decentralized	Autocratic
Perspective to Change	Change Adaptability	Change Sustainability
Culture	Leadership-Collaboration	Command-Control
Documentation	Low	Heavy
Emphasis	People-Oriented	Process-Oriented
Cycles	Numerous	Limited
Domain	Unpredictable/Exploratory	Predictable
Upfront Planning	Minimal	Comprehensive
Return on Investment	Early in Project	End of Project
Team Size	Small/Creative	Large

Table 2: Difference in Agile and Heavyweight Methodologies

The agile and heavyweight methodologies both have their strengths and weaknesses. People usually follow either one of these methodologies or follow their own customized methodology. There are major factors affecting methodology decision and selecting which is suitable for various conditions. These factors can be categorized into project size, people and risk.

7.1 Project Size

One of the limitations of agile methods is project size. The key elements of project size are project budget, duration and project team organization. The larger the team or more budget you need, the bigger the project is. Thus compiling more requirements, requiring more people and more coordination. Heavyweight methodologies support this by providing plans, documentation and processes for better communication and coordination across large groups. As seen in Figure 13 below, Alistair Cockburn one of the founders of

agile alliance, claims that for a given problem size, “fewer people are needed if a lighter methodology is used, and more people are needed if a heavier methodology is used,” and asserts that, “There is a limit to the size of problem that can be solved with a given number of people” [57].

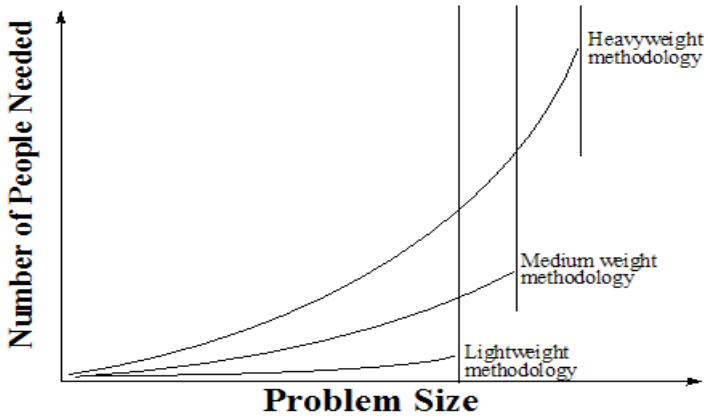


Figure 13: Problem Size and Methodology Affecting Staff [54]

The larger the team also affects the communication in a project and effectiveness per person. The figure below shows the communication load rising as the number of people increase causing the effectiveness per person to drop. Cockburn states that methodology is a matter of coordinating people and managing communication, therefore the level of methodology must rise as the number of people increases. This makes it more difficult to use agile methods with teams greater than 40 making heavyweight methodologies a preferred option for large teams. However, Ken Schwaber, a developer of SCRUM, disagrees with this stating, “...large teams can be decomposed into small sized teams using scrums of scrums” [13]. Project duration is another factor used for choosing a methodology. Heavyweight methodologies involve a lot of “time waste” outputs such as documentation, design documents, writing analysis etc. Concluding that when time is limited, using an agile methodology would be better.

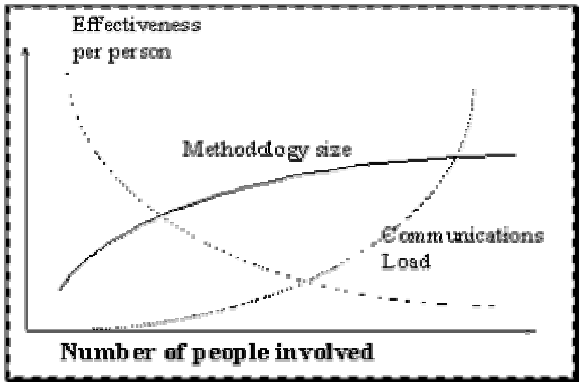


Figure 14: Effect of Project Size [58]

7.2 People Factor

Half of the agile manifesto values deal with human factors, “Individuals and interactions…” and “Customer collaboration...” [25]. Even NASA has concluded that technology and training are not the big factors, “The most effective practice is leveraging human potential”. Having skill and experienced people in a team is a key factor for agile methodologies. Encouraging domain experts to be part of the team gives developers rapid feedback on the implications to the user of their design choices. Customer adaptability is another great factor, the customer gets the power to check the progress and change the direction of the software development during each iteration. Gaining this level of commitment from the customer makes agile methodology a more attractive process than heavyweight.

The culture of an organization is an important factor when choosing a methodology. If an organization is solid, which is not responsive to changes and has many rules and procedures it cannot be successful using an agile methodology. Otherwise, if an organization is responsive or flexible, they must to adopt adaptability towards changes as their culture if they want to apply agile methods.

7.3 Risk Factors

The most important risk factors in the development of a software process are project criticality and responding to change. Agile methods are used in applications that can be built quickly and do not require extensive quality assurance. Critical, reliable, and safe systems are more suited to a heavyweight methodology. If a project is critical, all requirements must be well defined before the development of the software. Poor definition would result in more damage from undetected defects. Responding to change can be resolved using an agile method. Practices defined in agile methods allow for better handling the changes, such as constant feedback from customer and short iterative development.

Summarizing what was said above, Table 3 demonstrates the home grounds for agile and traditional methods, which includes the sets of conditions under which they are most likely to succeed. The more the project conditions differs from the home ground conditions the more the risk in using one approach over the other.

Project Characteristics	Agile discriminator	Heavyweight Discriminator
Primary objective	Rapid Value	High Assurance
Requirements	Largely emergent, rapid change, unknown	Knowable early, largely stable
Size	Smaller teams and projects	Larger teams and projects
Architecture	Designed for current requirements	Designed for current and foreseeable requirements
Planning and Control	Internalized plans, qualitative control	Documented plans, quantitative control
Customers	Dedicated, knowledgeable, collaborated, collocated onsite customers	As needed customer interactions, focused on contract provisions
Developers	Agile, knowledgeable, collocated, and collaborative	Plan-oriented; adequate skills access to external knowledge
Refactoring	Inexpensive	Expensive
Risks	Unknown risks, Major Impact	Well understood risks, Minor impact

Table 3: Agile and Heavyweight Discriminators [40]

8.0 Questionnaire

The information mentioned above is data and references from different sources. A couple of the sources were surveys conducted by Shine Technologies, CHAOS reports and Cutter Consortium. The following is a questionnaire (Appendix E) I developed to identify what methodologies software practitioners in government and commercial organizations in Perth follow to develop software for different sizes of projects. Information regarding their opinions on agile methodology and heavyweight methodology was also collected. A summary of the results and an analysis of the questionnaire are further discussed.

8.1 Questionnaire Format

The format of the questionnaire was developed in a way to make it easy and quick for the respondents to answer. The questions were all close-ended and were divided into four sections:

Organization Characteristics – This section deals with the type and size of the organization respondent resides. In addition, the organizations willingness to adopt new technologies and methods.

Methodology Questions – These questions rated the respondents' knowledge of both agile and heavyweight methodologies.

Software Development Questions – For this section the software development has been divided into three parts depending on the size of each project. These parts were Small-Scale project, Medium-Scale project and Large-Scale project which were categorized by the project time (in person-months). For each category of development, I aimed to discover the different heavyweight and agile methodologies used. Questions were also asked to determine which aspects of the agile and heavyweight methodology most appeal and do not appeal to the respondents in development. Furthermore, the respondents provided feedback on whether the adoption of agile and heavyweight methodologies had any effect on the project cost and software quality. Finally respondents gave their opinion on what they believe is a more suitable methodology for each scaled project and to what extent they would follow an agile technique whilst using a heavyweight methodology.

General Questions – These questions used to discover the respondent's position and if they wanted a report summarizing this study as an appreciation for their contribution.

8.2 Questionnaire Sample Size

The organizations chosen to complete the questionnaire ranged from a small, less than 10 full time software staff, to a large organization with over 100 full time staff. Overall there was a sample size of 15 respondents; the average time taken to complete the questionnaire was approximately 13 minutes.

8.3 Questionnaire Results

The analysis from the questionnaire results are discussed below and all findings are found graphically in Appendix F.

8.3.1 Results on Organization Characteristics and Respondents Knowledge

More than 50% of the respondents were from an Information Technology type organization, the other half were mainly from Government, Engineering and Other organizations. Majority of medical, education and some government organizations outsource larger software development projects to Information Technology companies. Furthermore, from the 15 organizations, 80% ranged equally between 10 full time staff to 100 full time staff and the remaining 20% were over 300 full time staff.

Of the respondents, 93% claim to have an understanding of average or higher of agile and heavyweight methods. 13 of the 15 respondents rate their knowledge of agile methods as average or extensive. 14 of the 15 respondents acknowledged that their knowledge of heavy methods was either extensive or very extensive. The respondents who had a less than average knowledge were given a lower rating to some questions because of their inadequate experience for this purpose.

When it comes to adopting new technologies and methods a massive 80% of respondents characterized their organization as either a market leader or a market follower. From the 80%, the majority used an agile methodology rather than a heavyweight. This observation supports our earlier statement concluding that majority of agile methodologies are only being accepted by Innovators and Early adopters according to Geoffrey Moore's Law of Technology Adoption Curve [28]. (*According to Moore, Innovators are those who pursue new concepts aggressively. Early Adopters are those who pursue new concepts very early in the lifecycle, followed by early majority and late majority. Laggards are those who simply don't want anything to do with the new approaches.[28]).* The remaining 20% described themselves conservative, ie a long time till they adopt agile methods.

8.3.2 Type of Methodology used

Extreme programming was by far the most popular agile methodology used. An interesting observation from the results collated showed that the next favorite was an in-house methodology developed by the organization. Moreover, the respondents choose two of the existing agile methodologies and mentioned aspects that are included in their in-house methodology. Another remark is if the respondent chose either Scrum or XP for a small-scale development, the same approach was reoccurring in the large-scale development as well. This is interesting, as it appears to indicate that the need to move to agile approaches has been independently derived by many different organizations.

As for heavyweight methodologies, approximately 55% of respondents used the Waterfall method, 22% chose the Unified process and 15% of the respondents had their own in-house methodology that they used for each of the different sized projects.

8.3.3 Effects of Agile methods on Software Quality and Cost compared to Heavyweight

In this section the respondent was supposed to give their opinion on whether they believe that taking on an agile method rather than a heavyweight method will affect the cost and quality of software. The results were not consistent with my expectations, this could be a result of several factors. For example, a respondent who has not used an agile method for medium and large scaled projects could be biased on his responses.

With regards to software costs, all respondents agreed that adopting an agile process in comparison to a heavyweight process for a small-scale project will result in a decrease in cost. However, when it came to medium and large-scale projects the costs started to increase for adopting an agile process rather than a heavyweight process. Approximately 53% of the respondents cast their votes in favor of the increased costs in the large-scale projects, 26% still believe it will decrease the cost and the rest of the respondents think it would have no effect to the cost.

When the issue of software quality was presented to the respondents, the majority of the respondents were inclined to say that the quality improvements follow heavyweight methodologies. More than 65% of respondents believed that there was a decrease in quality when using agile methods for medium and large scale projects.

The findings of the effects of agile methods on quality and cost do not support my earlier reading and observed study on this section. According to Shine Technologies 95% of the respondents stated that there was either no effect or a cost reduction on using agile methods. This shows that the set of respondents that I surveyed might not have used an agile methodology in a medium or large project. Furthermore 88% of Shine technologies respondents stated that the quality was better or significantly better using an agile method [44]. Agile methods should improve the quality of software as agile emphasizes on constant feedback from the customer onsite and constant communication between team

members. This process allows the customer to prioritize functionality needed and discard what is not. Figure 15 and 16 below show the results.

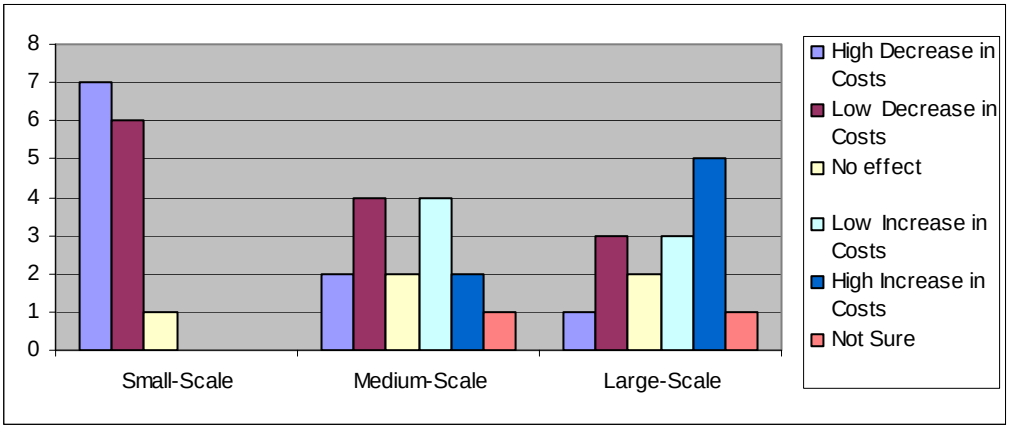


Figure 15: Effect of Agile Methods on Cost

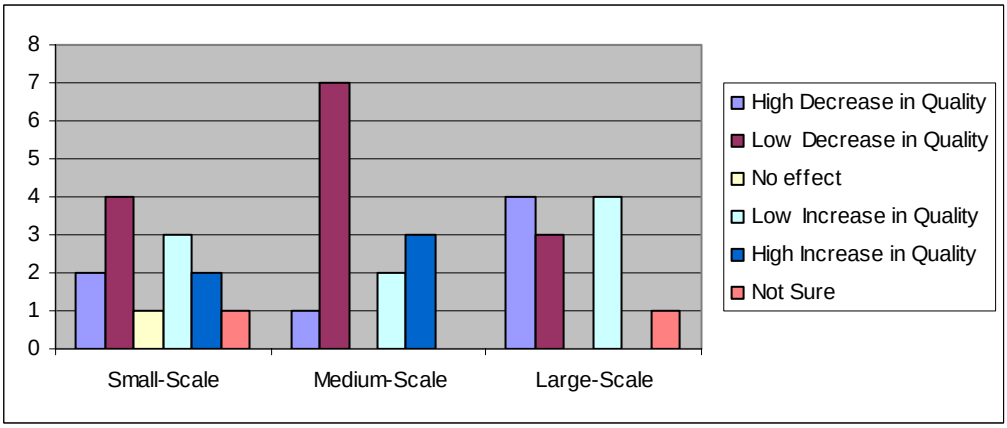


Figure 16: Effect of Agile Methods on Quality

8.3.4 Agile or Heavyweight for Software development

Despite the fact that 50% of respondents believed that agile methods decrease the quality of the software, nearly all respondents strongly agreed that they believe that agile methods are more compatible for small-scale projects. For medium-scale projects, the results were equally divided between agile and heavyweight methods. But for large-scale projects the respondents favored heavyweight methods. Why? Simply because heavyweight methods can easily plan the added complexity of running a large software team dispersed over multiple domains, functions, and continents in a more traditional organized way.

In the questionnaire, some respondents selected both agile and heavyweight methodology as a suitable candidate for medium-scale projects. This shows that organizations are slowly starting to adopt agile aspects into their heavyweight methodology for medium

sized projects and soon will start implementing for larger-scale software development. In the words of one of the respondent, “Getting an organization to switch to agile processes takes a lot of time and patience. Find a champion who is in a position to influence others.”

8.3.5 Likes and Dislikes of Agile and Heavyweight Methods

Whichever methodology the organization uses to develop their software there is always going to be a set of processes and methods. Therefore, I examined the respondents’ feedback on their likes and dislikes of particular aspects of agile and heavyweight methods for the different sizes of development.

For small scale projects there was no clear favorite agile aspect, Working Code versus Documentation had the most with 30% and the rest followed closely after. However for the medium and large-scale projects, Customer Relationship versus Contract Negotiation had more than 50% of respondents’ choice and stood out as the most appealing quality of agile methods. 40% were shared evenly between People versus Processes and Respond to change versus following a plan, while the remaining 10% chose Working Code over Documentation. The results are graphically shown below in Figure 17. As I discussed earlier human factors are an important aspect to project success. According to The Standish Group the top 2 most important factors in making a project successful have got to do with human factors [11].

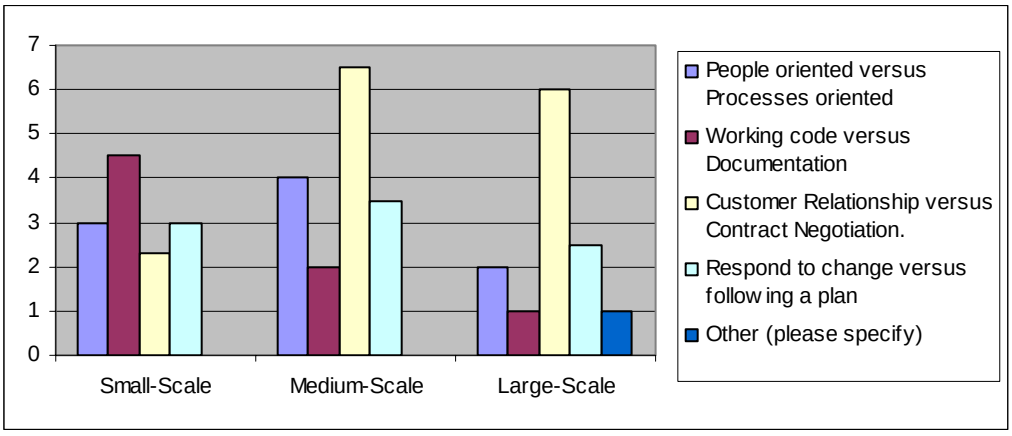


Figure 17: Aspects of Agile Methods Most Appealed by Respondents

In case of the Dislikes of Agile methods, Less Management Control was attributed as a downside for all kinds of software development especially small-scale and medium-scale projects. Similarly Lack of Project Structure is another major concern to all of the different scale projects. In fact, around 62% of the respondents sided with the fact that the main reason for the unacceptance of agile methods for large scale projects is the looseness of project structure in agile methods. In opposition, Bil Kleb from NASA Research Centre stated, “Don’t be fooled into thinking that Agile methodologies are not

very rigorous in terms of process. For example, XP, when used as prescribed (employing all 12 practices), is very strict indeed" [44].

For heavyweight methodologies Heavy Documentation was always a negative aspect for the different scaled projects. 50% of respondents did not like the fact that Heavy Documentation is needed for small scale projects. In addition, roughly 50% of the respondents of each medium and large scaled projects were of the opinion that they dislike the democratic management style of heavyweight methodologies. This type of management style does not allow the developers to be more creative and agile. According to Boehm this decreases their motivation, "Motivation has a larger effect on quality and productivity than any other factor" [22].

These outcomes confirm that these observations prove what was mentioned earlier in the report. Figure 18 and 19 below clarify the results of the respondents' results.

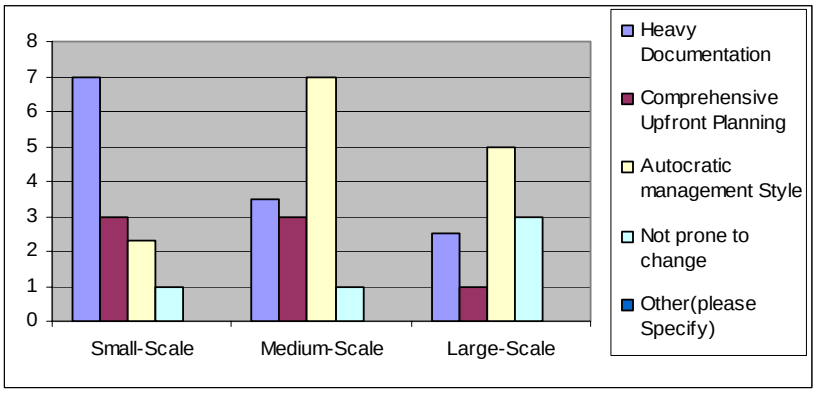


Figure 18: Dislikes of Agile Aspects

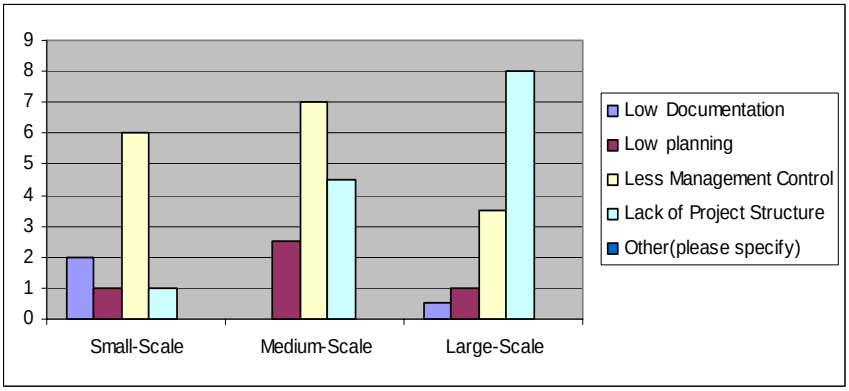


Figure 19: Dislikes of Heavyweight Aspects

At the same time I also collated information from respondents as to what were the major obstacles in practicing agile methods for different sizes of software projects. Figure 20 shows that lack of skilled people who can follow agile methodologies, was the major

factor in both small and medium scaled projects. Agilists agreed that a certain percentage of experienced people are needed in an agile method to bring the project along [22]. As Dan Mark states, "You need good, motivated people. Agile methodologies are hard work and require a very high degree of discipline to get it right."[15]. 60% of the respondents agreed that the major hurdle in using agile methods for large scale projects is project size and complexity. This was supportive to the argument I mentioned earlier in our report, as the project size increases the number of people rises, thus increasing communication. Agile methodologies rely heavily on communication, so large teams make it difficult to use agile methods. There is a clear inverse relationship between agile techniques and project complexity.

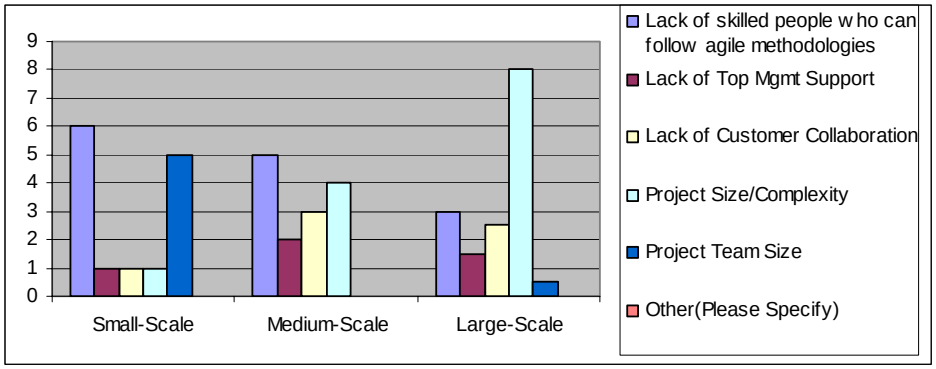


Figure 20: Common Problems in Agile Methods

Finally I measured the extent of using agile techniques for the different scaled projects. As seen below in Figure 21 none of the respondents follow all agile techniques. For small scale projects, averages of 58% of agile techniques are used when developing software. So for example if the respondent was using XP on a small scale project, 7 out of 12 XP techniques would be followed while the other 5 would either be changed or ignored to fit the project constraints and needs. The figure shows that fewer respondents are following agile techniques for medium and large scale projects, a weighted average of 34% and 22% respectively of the techniques were implemented in the projects. Therefore I could wrap up by saying that as the development of software increases, there is a corresponding decrease in the number of agile techniques used. Furthermore, respondents tend to use heavyweight methods when the increase of project size, complexity, team, architecture, scope, and risks occur.

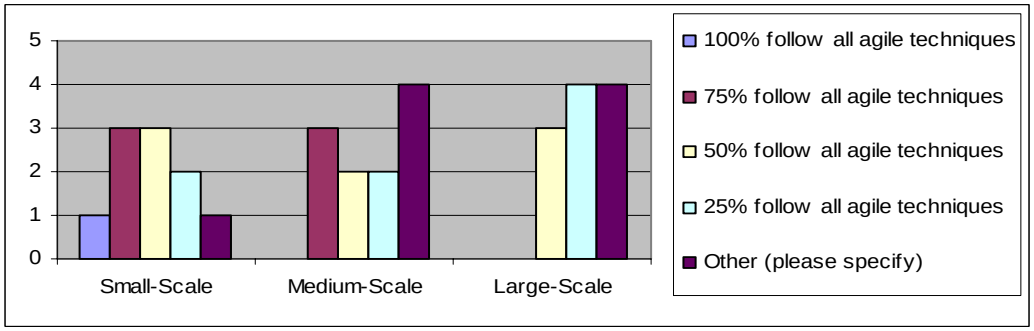


Figure 21: Extent of Agile Techniques

9.0 Conclusion and Future Work

In this dissertation, I described the different approaches to software development through heavyweight and agile methodologies. Furthermore, I initially criticized on both heavyweight and agile methodologies followed by the comparison. Further, I discussed the implementation of agile methods based on stories and anecdotal evidence of industrial teams experiencing success with agile methods. Moreover, I solicited and gathered feedback from software developers through a close-ended survey questionnaire. Although these provide valuable information about practical applications, empirical studies are needed for evaluating the effectiveness and the possibilities of using agile software development methods.

Throughout my research the dominance of heavy methodologies was apparent. This consisted of comprehensive planning, heavy documentation and extensive designs. The heavy thoughts that accompany them will be overtaken by the agile movement not far in the future. Heavyweight approaches will still have their need in large, long lived projects that have a special safety, reliability or security requirements. The defense industry is an example of this; however agile approaches are starting to be adopted in these areas. Tom DeMarco makes the analogy between military history and software development as each swing from the relative advantages of armor to those of mobility. He states: “In the field of IT, we are just emerging from a time in which armor (process) has been king. And now we are moving into a time when only mobility matters” [59].

Agile development is not defined by a small set of practices and techniques. From the set of success stories and anecdotal evidence we have come to believe that agile development defines a strategic capability, a capability to create and respond to change, a capability to balance flexibility and structure, a capability to draw creativity and innovation out of a development team, and a capability to lead organizations through turbulence and uncertainty. Heavyweight plan driven methodologies have a definite place for a less volatile era, where rigorous processes are applicable for a wide range of projects. However in this volatile environment and increasing uncertainty of what the customer wants, agile methods seem to be the dominant methodology. Companies want to create change for their competitors and respond quickly to market conditions. They plan, but are not blinded by those plans. They focus on delivering customer value, not adding up how many processes they have in place. They rough out blue prints (models) but concentrate on creating working software. They focus on individuals and their skills and on the intense interaction of development team members among themselves, customers and management.

As I mentioned earlier, the need for business to respond rapidly to the environment in an innovative, cost effective and efficient way is compelling the use of agile methods to developing software. According to the surveys done by Shine Technologies, the Standish Group and Cutter Consortium have shown that the percentage of companies using agile methods have increased each year. Just as mobile phones have reduced the need for

telephone landlines agile methods are reducing the need for heavyweight methodologies. The future of agile methodologies seems very dominant. In general, there are some aspects of software development project can benefit from an agile approach and others can benefit from a more predictive traditional approach. When it comes to methodologies, each project is different. One thing is clear: that there is no “one-size-fits-all” solution.

10.0 References

[1] D. Marks, "Development Methodologies Compared", *N CYCLES software solutions*, December 2002 , www.ncycles.com , Accessed on 2/2/2005

[2] B. Grady, C. Robert, J. Newkirk, *Object Oriented Analysis and Design with Applications*, 2nd edition, Addison Wesley Longman, 1998

[3] P. Kruchten, "What is Rational Unified Process?", *The Rational Edge*, http://www.therationaledge.com/content/jan_01/f_rup_pk.html Accessed 2/2/2005

[4] C. Larman, *Agile & Iterative Development: A Manager's Guide*. Addison-Wesley, 2004.

[5] B. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, May 1998

[6] M. Fowler, "The New Methodology," http://www.martinfowler.com/articles/newMethodology.html Accessed on 12/12/2004

[7] *Oxford English Dictionary*, http://www.dictionary.oed.com, Accessed 20/4/2005

[8] K. Beck, Embracing change with Extreme Programming. *IEEE Computer*, Vol. 32, Issue 10 October 1999.

[9] K. Beck, Extreme Programming explained: Embrace change. *Reading, Mass., Addison-Wesley, Nov16, 2004*

[10] L. A. Williams, "The XP Programmer: The Few-Minutes Programmer", *IEEE Software*, pp. 16-20, May/June 2003

[11] K.Mar and K.Schwaber, "Experiences of using Scrum with XP", http://www.controlchaos.com/XPKane.htm, Accessed on 2/2/2005

[12] L. Rising and N. S. Janoff, The Scrum software development process for small teams, *IEEE Software*, Issue 17, pp. 26-32 , 2000

[13] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Upper Saddle River, NJ, Prentice – Hall, 1st Edition, Oct 2001

[14] Advanced Development Methods, Inc., "What is Scrum?", http://www.controlchaos.com. Accessed on 2/4/2005

[15] S.R. Palmer and J.M. Felsing, *A Practical Guide to Feature-Driven Development*. Upper Saddle River, NJ, Prentice-Hall, 2002

[16] J. De Luca, "Feature Driven Development Overview," http://www.nebulon.com/articles/fdd/download/fddoverview.pdf, Accessed 20/4/2005

[17] Dynamic System Development Method Consortium, "DSDM Tour", http://www.dsdm.org, Accessed 20/2/2005

[18] J. Stapleton, *Dynamic systems development method – The method in practice*. Addison Wesley 1997.

[19] J. A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Addison Wesley 2000.

[20] W. Cunningham, "Agile Manifesto." http://www.agilemanifesto.org/, Accessed on 10/7/2004

[21] S. W. Ambler, "Duking it out", *Software Development*, July 2002

[22] First eWorkshop on Agile Methods, Centre for Experimental Software Engineering Maryland, April 8 2002

[23] P. Wendorff, *An Essential Distinction of Agile Software Development Processes Based on Systems Thinking in Software Engineering Management.* Addison Wesley; page 218.

[24] J. Highsmith and A. Cockburn, "Agile Software Development: The Business of Innovation", *IEEE Computer*, http://www.jimhighsmith.com/articles/IEEEArticle1Final.pdf Accessed on 10/10/2004

[25] J. Highsmith and A. Cockburn, "Agile Software Development: The People Factor", *IEEE Computer*, http://www.jimhighsmith.com/articles/IEEEArticle2Final.pdf Accessed on 10/10/20054

[26] J. Highsmith, *Agile Software Development Ecosystem*. Addison Wesley, 2002

[27] M. Fowler, "Is Design Dead?", *Software Development*, http://www.martinfowler.com/articles/designDead.html Accessed on 2/2/2005

[28] L. Williams and A. Cockburn, "Agile Software Development: It's about Feedback and Change," *IEEE Computer*, June 2003, pp. 39-43

http://fc-md.umd.edu/projects/Agile/Summary/Summary2.htm Accessed on 2/2/2005

[29] Second eWorkshop on Agile Methods, Centre for Experimental Software Engineering Maryland, June 19 2002 http://fc-md.umd.edu/projects/Agile/secondeworkshop/summary2ndeworksh.htm Accessed on 2/2/2005

[30] B. Barry and P. Philip, "Understanding and Controlling Software Costs", *IEEE Transactions on Software Engineering*, Vol. 14 No 10, October 1988

[31] The Standish Group International, "The CHAOS Report ," http://www.standishgroup.com/sample_research/chaos_1994_1.php Accessed on 2/2/2005

[32] C. Jones, *Applied Software Measurements*. McGraw Hill,1997

[33] M. Snider, "Simplicity Now!," Oracle Publishing, http://www.oracle.com Accessed on 2/12/2004

[34] K. Eisenhardt and D. Sull, "Strategy as Simple Rules," *Harvard Business Review*, Vol. 79, pp. 107-116, Jan 2001

[35] M. Popendieck, "Lean Programming," http://www.poppendieck.com/lean.htm Accessed on 2/2/2005

[36] A. MacCormack, "Product-Development Practices that Work: How Internet Companies Build Software," *MIT Sloan Management Review*, Vol.42, Winter 2001

[37] M. Buckingham and D. Clifton, "Now, Discover Your Strengths", *The Gullop Organization*, 2001

[38] A. Cockburn, "Characterizing People as Non-Linear, First-Order Components in Software Development," http://alistair.cockburn.us/crystal/articles/cpanfocisd/characterizingpeopleasnonlinear.html Accessed on 2/2/2005

[39] L. Constantine, "Methodological Agility," *IEEE Software Development*, June 2001, pp.67-69

[40] B. Boehm, "Get Ready for Agile Methods, with Care," *IEEE Software Development*, January 2002, pp.64-69

[41] S. Ambler, "When Does(n't) Agile Modeling Make Sense?", http://www.agilemodeling.com/essays/whendoesamwork.htm#WontWork ,Accessed on 2/2/2005

[42] P. Cauwenberghe, “Refactoring or Upfront design?” http://www.agilealliance.org/articles/articles/Chapter12-VanCauwenberghe.pdf Accessed on 2/2/2005

[43] A. Cockburn, “Reexamining the Cost of Change Curve” http://www.xprogramming.com/xpmag/cost_of_change.htm Accessed on 2/2/2005

[44] Shine Technologies, “Agile Methodologies Survey Results” http://www.shinetech.com/display/www/Extreme+success+with+Agile#attachments Accessed on 2/2/2005

[45] S. W. Ambler, “Answering the ‘Where is the Proof That Agile Methods Work’ Question”, http//www.agilemodeling.com/essays/proof.htm , Accessed 2/2/2005

[46] L. Copeland, “Extreme programming moves slowly into the enterprise”, *Computerworld*, October 2001

[47] G. Moore, *Crossing the Chasm*. HarperBusiness, Revised Edition 2002

[48] K. Beck, “Embracing Change with Extreme Programming”, *IEEE Software*, pp.70 -77, October 1999

[49] J. Highsmith, “What Is Agile Software Development?”, *The Journal of Defense Software Engineering*, pp.4-9, October 2002

[50] L. Copeland, “Caterpillar Digs Into Agile Development”, *Computerworld* , January 2002

[51] C. Sliwa , “Agile Programming techniques spark interest”, *Computerworld*, March 2002.

[52] R. Charette, “The Decision is in:Agile Versus Heavy Methodologies”, *Cutter Consortium*, Vol. 2 No.19, March 2002

[53] M. C. Paulk, “Extreme Programming from a CMM Prespective,” *IEEE Software*, pp.19-26, Nov/Dec 2001

[54] S. Moore and L. Barnett , “Offshore Outsourcing And Agile Development” , *Forrester Best Practices*, September 20, 2004 .

[55] M. Simons, “Internationally Agile”, *InformIT*, March 15, 2002.

[56] M. Fowler, “Using an Agile Software Process with Offshore Development”, http://www.martinfowler.com/articles/agileOffshore.html Accessed on 2/2/2005

[57] A. Cockburn, “The Methodology Space”, http://alistair.cockburn.us/crystal/articles/ms/methodologyspace.htm Accessed on 2/2/2005

[58] A. Cockburn, “A Methodology Per Project”, http://alistair.cockburn.us/crystal/articles/mpp/methodologyperproject.html Accessed on 2/2/2005

[59] K. Beck and M. Fowler, *Planning eXtreme Programming*. Boston Addison Wesley, 2001

APPENDIX A

Original Honours Proposal

Title: Analysis of various methodologies used in developing information systems within organizations

Author: Mohamed Awad

Supervisor: Mr. Alex Reid and Mr. Terry Woodings

Background

One of the most valuable asset of modern corporations is information but development of Information systems faces many problems. Problems consist of low productivity, a large number of failures and an inadequate alignment of Information Systems with business needs. The first problem low productivity simply demands for building new or improved Information systems have increased faster than our ability to develop them. Second the numbers of failures are due to economical mismatches, such as budget and schedule overruns. An IBM's software project survey showed that 55% of the software developed cost more than projected, 68% took longer to complete than predicted, and 88% had to be substantially redesigned.

Third problem from a business point of view, the link between Information systems and organizational performance and strategies has been shown to be doubtful. Information system development is continually challenged by the dynamic nature of business together with the ways the business activities are organized and supported by information systems.

All the above problems are further aggravated by the increasing complexity and size of software products. That is why companies are facing challenges in developing new strategies for developing Information Systems as well as in finding supporting tools and ways of working. One widely acknowledged approach to solve these problems has been to improve and apply systematic guidelines and procedures for developing IS. The goal of method development is to build up collective experience of IS development and utilize it to craft systematic development practices. As a result methodical approaches are expected to lead to more acceptable and successful solutions and to a better managed development process. We currently find hundreds of methods of developing Information system and new or improved methods are being introduced continuously.

In the past there used to be one main type of methodology used called the heavyweight methodologies which consists of comprehensive planning, a lot of documentation and

extensive design. New methodologies called the lightweight methodologies, aka agile modeling, subsumes individuals over processes, working software over documentation, collaboration over negotiation, and responding to change over following a plan.

Information system development is defined as a change process taken with respect to object systems in a set of environments by a development group using tools and an organized collection of techniques to achieve or maintain an objective.

Aim

During the course of the project I will endeavor to review and critique on the various methodologies used in developing an Information system and work out which methodology is best to use when developing a specific type of system. In order to do this I would have to follow these steps:

1. First I would conduct a general review of Development of Information System. This would include the concepts, issues and practices used to develop a system.

2. Literature review of both heavyweight and agile methodologies. In the heavyweight review I would discuss a few of the heavyweight methods such as Waterfall, Spiral and UDP. Similarly in the lightweight methodologies I would make a brief description of agile methods such as extreme programming, SCRUM, DSDM, FDD, and Adaptive software development. Carry out an evaluation of the strengths and weakness of all identified methodologies

3. Next I would critique on both the heavyweight and lightweight methodologies and provide a comparative analysis on both – the objective, scope, resources, architecture, size and requirements.

4. Make a questionnaire to get feedback from software developers of different companies in Perth and to find out what methodologies are used in different sized companies. A questionnaire is a viable research method because it is an easier and quicker respondent to answers. The questions are close ended so this provides a flexibility to code and statistically analyse response choices.

5. Provide this questionnaire to companies in Dubai to get an international insight of what they use in the Middle East and compare it to the results from Perth. Dubai is becoming the centre of trade of companies from all around the world so the methodologies used there would be from companies coming from countries like the United States, Canada and all around Europe. This would allow us to see which of the two, heavyweight or Agile, is used in international companies.

6. Identify and collect quantitative data on use, efficiency, time, cost success rates.

Finally after completing all those steps I would be able to draw conclusions, recommendations and principles about the state of and the process of information system development.

Method

As mentioned above I would be dividing this project in a number of steps.
The main task in the first stage is to generalize the methods and characteristics of heavyweight and agile methodologies. Plus provide an insight of the development of Information system such as the concepts, issues and practices behind software development. Then wrap up the literature review by giving our analysis and compare and contrast heavyweight and agile methods according to different project characteristics, such as objective, scope, resources, architecture, size, and requirements.
The second stage is to make a closed-ended questionnaire to give companies in Perth to gather and analyse information of methodologies that are currently used.
As well I would be traveling to Dubai during the summer where I would be distributing this questionnaire to companies there. This would provide me with an insight with the methodologies used internationally and I could compare it with ones used in Perth.
In the third and final stage I would wrap up all the information to provide an analysis on the different methodologies used internationally and used here in Perth. With all these three stages I would be able to recommend and conclude on each methodology and show which is best suited to what kind of Information System.

Plan

The following is a more detailed plan of task and milestones that I need to follow:

Task	Proposed Deadline Semester 2
Research Proposal	Week 4
Read Literature on Information System Development	Week 5
Write up summary report on Information System Development	Week 5
Read Literature on methodologies	Week 6
Read literature on Agile methodologies	Week 7
Read literature on Heavyweight methodologies	Week 7
Read journals on Information System Development	Week 8
Read Journals on Agile & Heavyweight	Week 8

Methodologies	
Compare Agile methodologies	Week 9
Compare Heavyweight methodologies	Week 10
Write up summary report on Agile methods	Week 11
Write up summary report on Heavyweight methods	Week 12
Provide a comparison on both methodologies	Week 13
Write up a Questionnaire for software developers, go through it with supervisors	Week 14
Send questionnaire to companies in Dubai	*Summer Break*
Visit software developing companies in Dubai	
Make an interview with a couple companies in Dubai	
Analysis the methodologies used in Dubai	
Continuing reading more reports and journals on software development methodology in Middle East	
Provide an insight of the information read in Dubai compared to Perth	
Task	**Semester 1 2005**
Revise with supervisor the information provided from Dubai	Week 1
Send Questionnaires to companies in Perth	Week 2
Make interviews with Perth software company	Week 2
Write up report analysing questionnaire results	Week 3
Provide questionnaire results and analysis to companies that did the questionnaire	Week 3
Compare Results from Perth with results in Dubai (the difference in methodologies used)	Week 4
Write up all conclusions of information gathered and provide pie charts, tables etc..	Week 5
Finish writing up dissertation	Week 6
Draft dissertation due to project supervisor(s)	Week 7 Thursday 21 April
Draft dissertation available for collection from project supervisor(s)	Week 9 Thursday 5 May
Seminar title and abstract due to 4th Year Coordinator	Week 10 4pm Thursday 12 May

Final dissertation due to 4th Year Coordinator	Week 12 4pm Thursday 26 May
Seminar presented to seminar marking panel	Week 13 31 May - 2 June
Poster due	Week 13 4pm Thursday 2 June
June TBA	Marked dissertation available for collection from 4th Year Coordinator
June TBA	Corrected dissertation due to 4th Year Coordinator

APPENDIX B

Feature Driven Development Roles and Responsibilities

The six key roles in a FDD project are [9]:

Project Manager – The project manager is the administrative and financial leader of the project. In FDD, the project manager has the ultimate say on the scope, schedule, and staffing of the project.

Chief Architect – The chief designer is responsible for the overall design of the system and running the workshop design sessions held wit the team. They also have the final decision on the all designs.

Chief Programmer – The chief programmer is an experienced developer, who participates in the requirements analysis and design of the projects. The chief programmer is in charge of selecting the features from the feature set to be developed in each iteration of the development process.

Class Owners – Class owners work under the guidance of the chief programmer in the task of designing, coding, testing and documenting. They are responsible for the development of the class they have been assigned to be the owner of.

Domain Expert – The domain expert may be a user, a client, a sponsor, a business analyst, or a mixture of these. Their task is to possess the knowledge of how the different requirements for the system under development should perform. Domain experts pass this knowledge to the developers in order to ensure that the developers deliver a competent system.

APPENDIX C

Lean Manufacturing and Total Quality Management (TQM) Rules

The basic practices of Deming's TQM movement and Ohno's Lean Production can be summed up in these 10 points [34, 35]:

1. Eliminate waste.
2. Minimize inventory.
3. Maximize flow.
4. Pull from demand.
5. Meet customer requirements.
6. Do it right the first time.
7. Empower workers.
8. Ban local optimization.
9. Partner with suppliers.
10. Create a culture of continuous improvement

APPENDIX D

Capability Maturity Model standards [53]

The Software CMM key process areas and their purposes

Key process area	Purpose
	Maturity Level 2: Repeatable
Requirements management	Establish a common understanding between the customer and software project team about the customer's requirements.
Software project planning	Establish reasonable plans for software engineering and overall project management.
Software project tracking and oversight	Provide adequate visibility into actual progress so that management can act effectively when the software project's performance deviates significantly from the software plans.
Software subcontract management	Select qualified software subcontractors and manage them effectively.
Software quality assurance	Provide management with appropriate visibility into the product and the software process.
Software configuration management	Establish and maintain the integrity of software products throughout the project's software life cycle.
	Maturity Level 3: Defined
Organization process focus	Establish organizational responsibility for software process activities that improve the organization's overall software process capability.
Organization process definition	Develop and maintain a usable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term organizational benefits.
Training program	Develop individuals' skills and knowledge so that they can perform their roles effectively and efficiently.
Integrated software management	Integrate the software engineering and management activities into a coherent, defined software process based on the organization's standard software process and related process assets.
Software product engineering	Consistently use a well-defined engineering process that integrates all the software engineering activities to produce correct, consistent software products effectively and efficiently.
Intergroup coordination	Establish a way for the software engineering group to participate actively with other engineering groups so that the project can effectively and efficiently satisfy customer needs.
Peer reviews	Remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software products and the preventable defects.
	Maturity Level 4: Managed
Quantitative process management	Quantitatively control the performance of the software project's process. Software process performance represents the actual results achieved from following a software process.
Software quality management	Quantify the quality of the project's software products and achieve specific quality goals.
	Maturity Level 5: Optimizing
Defect prevention	Identify the cause of defects and prevent them from recurring.
Technology change management	Identify new technologies (such as tools, methods, and processes) and introduce them into the organization in an orderly manner.
Process change management	Continually improve the organization's software processes with the goal of improving software quality, increasing productivity, and decreasing the product-development cycle time.

[table taken from 53]

APPENDIX E

Software Development Methodology Questionnaire

School of Computer Science & Software Engineering

Title: Analysis of various methodologies in use for developing information system within organizations

Author: Mohamed Awad

Supervisors: Mr. Alex Reid and Mr. Terry Woodings

Questionnaire Confidentiality
This Questionnaire is used for a thesis done in the University of Western Australia. The information you give will be completely confidential and at all times, data will be presented in such a way that your identity cannot be connected with specific published data.

For more information please contact myself, Mohamed Awad:
Email: awadm01@tartarus.uwa.edu.au
Mobile: 0401746004

Or my supervisors
Mr. Terry Woodings
School of Computer Science & Software Engineering
The University of Western Australia
M002, 35 Stirling Highway
Crawley, Western Australia, 6009

 Phone +61 8 6488 2618

Professor Alex Reid
School of Computer Science & Software Engineering
The University of Western Australia
35 Stirling Highway
Crawley, Western Australia, 6009

 Phone +61 8 9345 0440

A. Organization Characteristics

1. What type of business or organization are you employed in?

☐Information Technology
☐Telecommunications
☐Engineering
☐Medical
☐Education
☐Government
☐Other

2. Approximately how many software professionals are employed by your organization?

☐Less than 10 full time staff
☐10 to 20 full time staff
☐21 to 50 fulltime staff
☐51 to 99 full time staff
☐100 to 300 full time staff
☐Other

3. Do you use any Software Capability Quality standards?
(such as ISO 9000, SPICE, CMMI)

☐Yes
☐ No

4. When it comes to adopting new technologies and methods, your company is

☐Market Leader (expands their total market by adopting new technology)
☐Market Follower (happy to adopt the technology after the leader)
☐Conservative (only follows when technology proven)
☐Static (does not accept new technologies)

B. Methodology Questions

5. How would you rate your knowledge of Agile Methodologies*?

☐Very Limited

☐Limited

☐Average

☐Extensive

☐Very Extensive

6. How would you rate your knowledge of Heavyweight Methodologies?**

☐Very Limited

☐Limited

☐Average

☐Extensive

☐Very Extensive

**Agile Methodologies: employ short iterative cycles, and rely on tacit knowledge within a team as opposed to documentation .e.g. XP programming, SCRUM*

** *Heavyweight Methodologies are considered the traditional way to develop software using a requirement-design-build paradigm with standard, well- defined processes.e.g. Waterfall*

C. Software Development Questions

For this section software development has been divided into three parts depending on the size of each project.

Small-Scale Project
Project time = less than 6 person months

Medium-Scale Project
Project time = 6 person months – 4 person years

Large-Scale Project
Project time = more than 4 person years

---

7. Which Agile Methodology do you mostly use for different kinds of Software development? *Please specify if more than one*

Small-Scale	Medium-Scale	Large-Scale
☐Extreme Programming ☐Scrum ☐DSDM ☐Feature Driven ☐Adaptive Software Dev ☐Other(please specify)	☐Extreme Programming ☐Scrum ☐DSDM ☐Feature Driven ☐Adaptive Software Dev ☐Other(please specify)	☐Extreme Programming ☐Scrum ☐DSDM ☐Feature Driven ☐Adaptive Software Dev ☐Other(please specify)

8. Which Heavy methodology do you mostly use for different kinds of Software development?

Small-Scale	Medium-Scale	Large-Scale
☐Waterfall ☐Spiral ☐Unified Process ☐Other(please specify)	☐Waterfall ☐Spiral ☐Unified Process ☐Other(please specify)	☐Waterfall ☐Spiral ☐Unified Process ☐Other(please specify)

9. Which of the listed aspects of Agile Methodologies most appeal to you compared with Heavyweight Methodologies, for the 3 sizes of software development project?

Small-Scale	Medium-Scale	Large-Scale
☐People oriented versus Processes oriented ☐Working code versus Documentation ☐Customer Relationship versus Contract Negotiation. ☐Respond to change versus following a plan Other (please specify)	☐People oriented versus Processes oriented ☐Working code versus Documentation ☐Customer Relationship versus Contract Negotiation. ☐Respond to change versus following a plan Other (please specify)	☐People oriented versus Processes oriented ☐Working code versus Documentation ☐Customer Relationship versus Contract Negotiation. ☐Respond to change versus following a plan Other (please specify)

10. Which aspect of agile methodologies, do you dislike the most for different kinds of software development?

Small-Scale	Medium-Scale	Large-Scale
☐Low Documentation ☐Low planning ☐Less Management Control ☐Lack of Project Structure Other(please specify)	☐Low Documentation ☐Low planning ☐Less Management Control ☐Lack of Project Structure Other(please specify)	☐Low Documentation ☐Low planning ☐Less Management Control ☐Lack of Project Structure Other(please specify)

11. Which aspect of Heavy methodologies, do you dislike the most for different kinds of software development?

Small-Scale	Medium-Scale	Large-Scale
☐Heavy Documentation ☐Comprehensive Upfront Planning ☐Autocratic management Style ☐Not prone to change Other(please Specify)	☐Heavy Documentation ☐Comprehensive Upfront Planning ☐Autocratic management Style ☐Not prone to change Other(please Specify)	☐Heavy Documentation ☐Comprehensive Upfront Planning ☐Autocratic management Style ☐Not prone to change Other(please Specify)

12. How do you believe that the cost of employing Agile Methodologies compares with Heavyweight Methodologies for the 3 sizes of software development project? *(select one in each category)*

Small-Scale	Medium-Scale	Large-Scale
☐High Decrease in Costs ☐Low Decrease in Costs ☐No effect ☐Low Increase in Costs ☐High Increase in Costs ☐Not Sure	☐High Decrease in Costs ☐Low Decrease in Costs ☐No effect ☐Low Increase in Costs ☐High Increase in Costs ☐Not Sure	☐High Decrease in Costs ☐Low Decrease in Costs ☐No effect ☐Low Increase in Costs ☐High Increase in Costs ☐Not Sure

13. Do you believe that taking on of agile methodologies rather than Heavyweight methodologies have any effect on Software Quality for different levels of development?

Small-Scale	Medium-Scale	Large-Scale
☐High Decrease in Quality ☐Low Decrease in Quality ☐No effect ☐Low Increase in Quality ☐High Increase in Quality ☐Not Sure	☐High Decrease in Quality ☐Low Decrease in Quality ☐No effect ☐Low Increase in Quality ☐High Increase in Quality ☐Not Sure	☐High Decrease in Quality ☐Low Decrease in Quality ☐No effect ☐Low Increase in Quality ☐High Increase in Quality ☐Not Sure

14. What do you believe is the most common problem experienced while practicing agile methodologies for different kinds of software development?

Small-Scale	Medium-Scale	Large-Scale
☐Lack of skilled people who can follow agile methodologies ☐Lack of Top Mgmt Support ☐Lack of Customer Collaboration ☐Project Size/Complexity ☐Project Team Size Other(Please Specify)	☐Lack of skilled people who can follow agile methodologies ☐Lack of Top Mgmt Support ☐Lack of Customer Collaboration ☐Project Size/Complexity ☐Project Team Size Other(Please Specify)	☐Lack of skilled people who can follow agile methodologies ☐Lack of Top Mgmt Support ☐Lack of Customer Collaboration ☐Project Size/Complexity ☐Project Team Size Other(Please Specify)

15. What is the Average Size of teams that work on Software Development in each project category, in your organization?

Small-Scale	Medium-Scale	Large-Scale
☐2-15 team members ☐15-50 team members ☐50-200 team members ☐More than 200 team members	☐2-15 team members ☐15-50 team members ☐50-200 team members ☐More than 200 team members	☐2-15 team members ☐15-50 team members ☐50-200 team members ☐More than 200 team members

16. What do you believe is the most suitable methodology for the different kinds of Software Development?

Small-Scale	Medium-Scale	Large-Scale
☐Agile ☐Heavy ☐Not Sure Other(please specify)	☐Agile ☐Heavy ☐Not Sure Other(please specify)	☐Agile ☐Heavy ☐Not Sure Other(please specify)

17. Do you use any other methodology other than agile and heavy methodologies for different kinds of software development?

Small-Scale	Medium-Scale	Large-Scale
☐Yes (please specify) ☐No	☐Yes (please specify) ☐No	☐Yes (please specify) ☐No

18. To what extent, do you follow different kinds of agile techniques for different kinds of software development?

Small-Scale	Medium-Scale	Large-Scale
☐100% follow all agile techniques ☐75% follow all agile techniques ☐50% follow all agile techniques ☐25% follow all agile techniques Other (please specify)	☐100% follow all agile techniques ☐75% follow all agile techniques ☐50% follow all agile techniques ☐25% follow all agile techniques Other (please specify)	☐100% follow all agile techniques ☐75% follow all agile techniques ☐50% follow all agile techniques ☐25% follow all agile techniques Other (please specify)

D. General Questions

Which of the following best describes your position in the organization?

☐Programmer / Developer
☐Analyst
☐Software Architect
☐Software Engineering
☐Consultant
☐Project Manager
☐Executive
☐Other (please specify)

Would you like to receive the report summarizing this study?

☐Yes (*please provide email address below*)

☐No

Would you like to provide details of other people who might be suitable to answer this questionnaire? *(even if only company name)*

Name	Company	Contact	Email

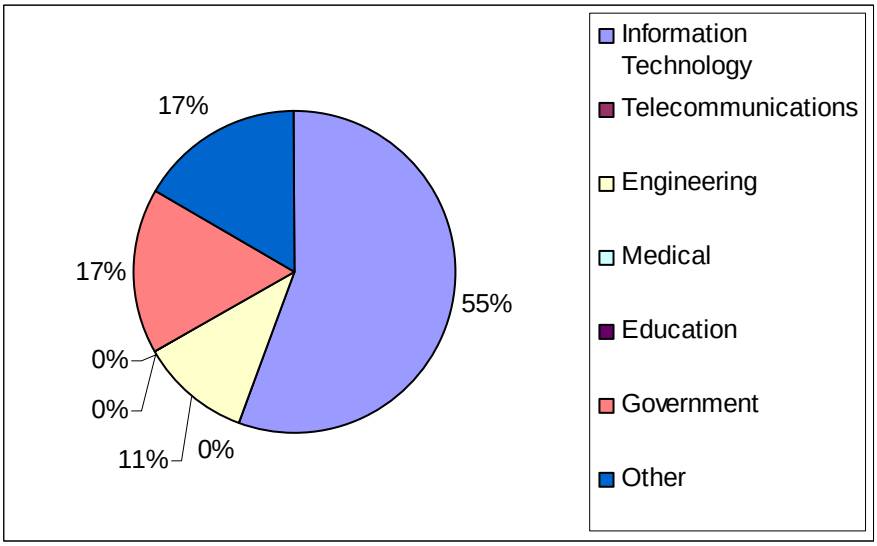
THANK YOU FOR TAKING THE TIME TO FILL OUT THE QUESTIONNAIRE. It is really appreciated as it would be truly helpful for my thesis work. Again all information is kept confidential and no organizations names would be mentioned in any published work.
Have a nice day!

APPENDIX F

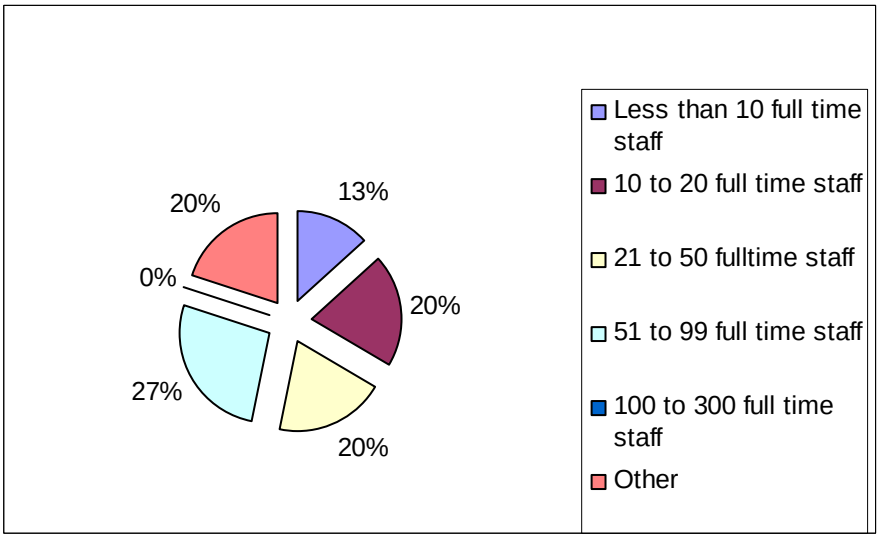
Questionnaire Results for All Samples

These are the results of each questions of the questionnaire

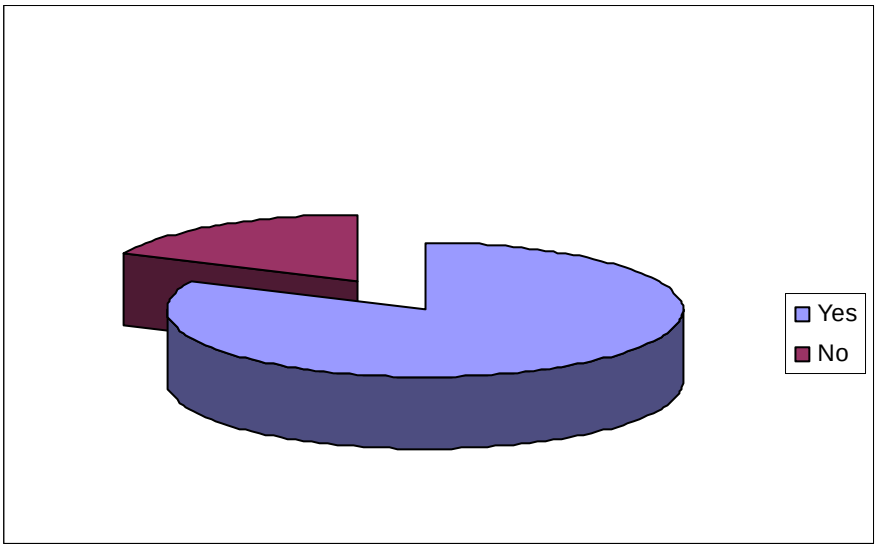
1. **What type of business or organization are you employed in?**



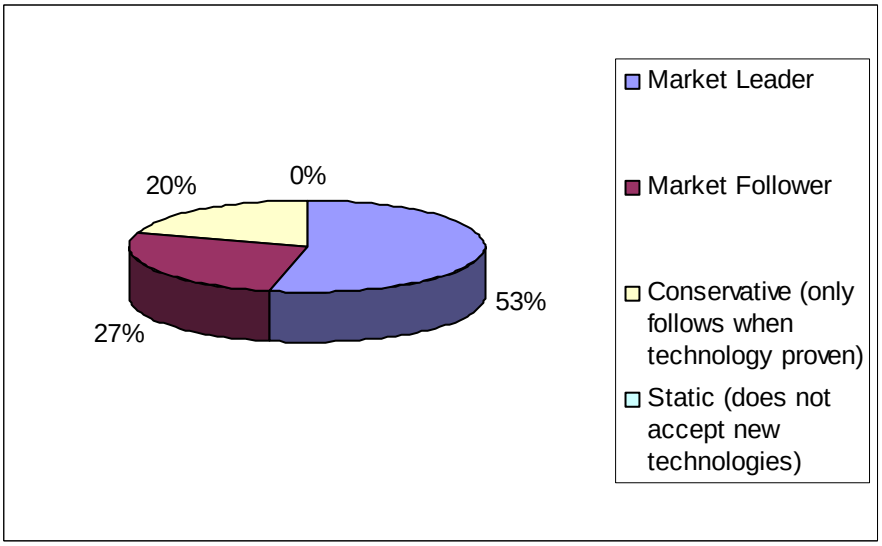
2. **Approximately how many software professionals are employed by your organization?**



3. Do you use any Software Capability Quality standards?
(such as ISO 9000, SPICE, CMMI)



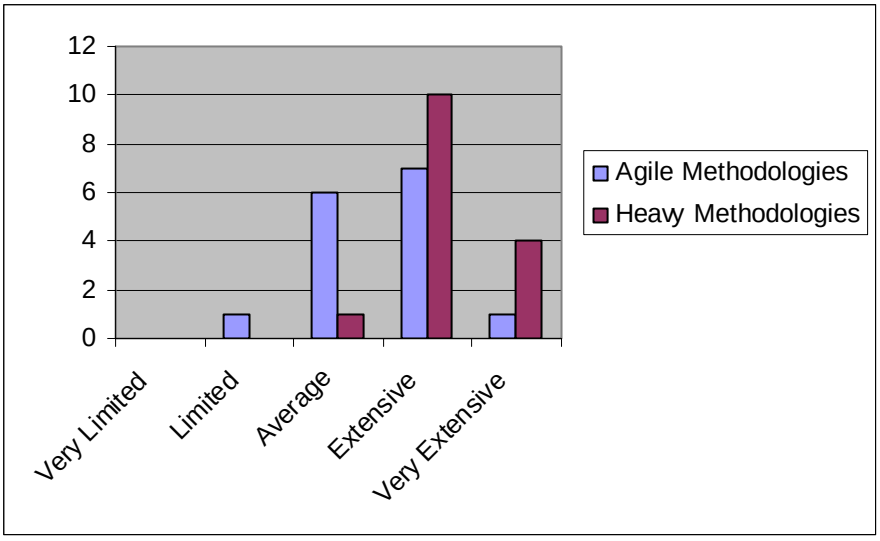
4. When it comes to adopting new technologies and methods, your company is



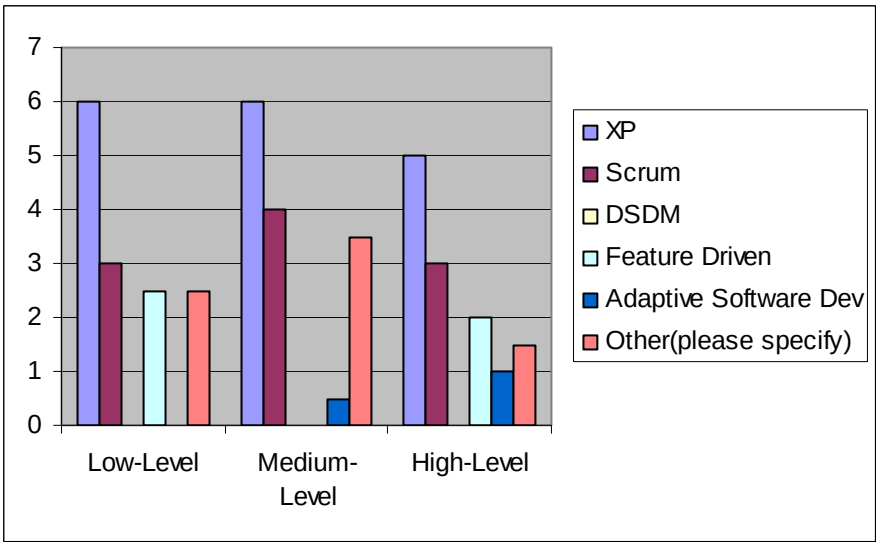
B. Methodology Questions

5. How would you rate your knowledge of Agile Methodologies?

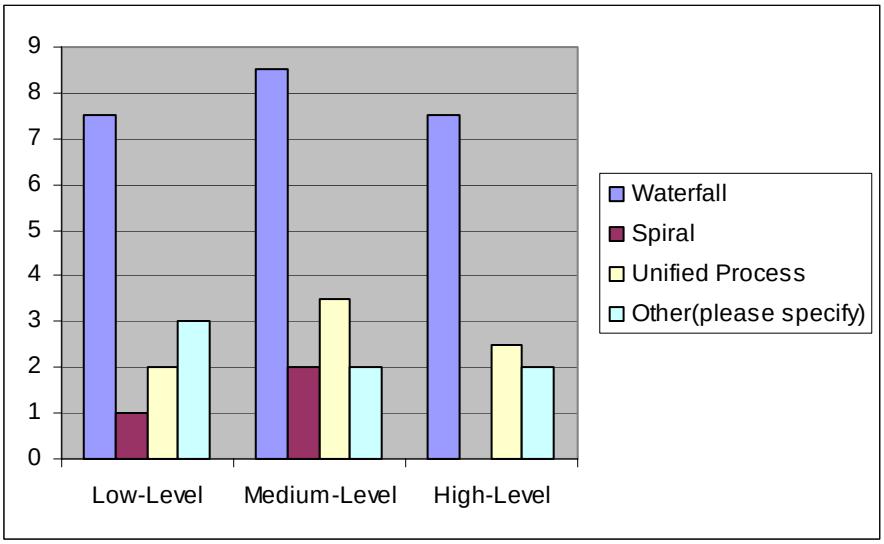
6. How would you rate your knowledge of Heavyweight Methodologies?



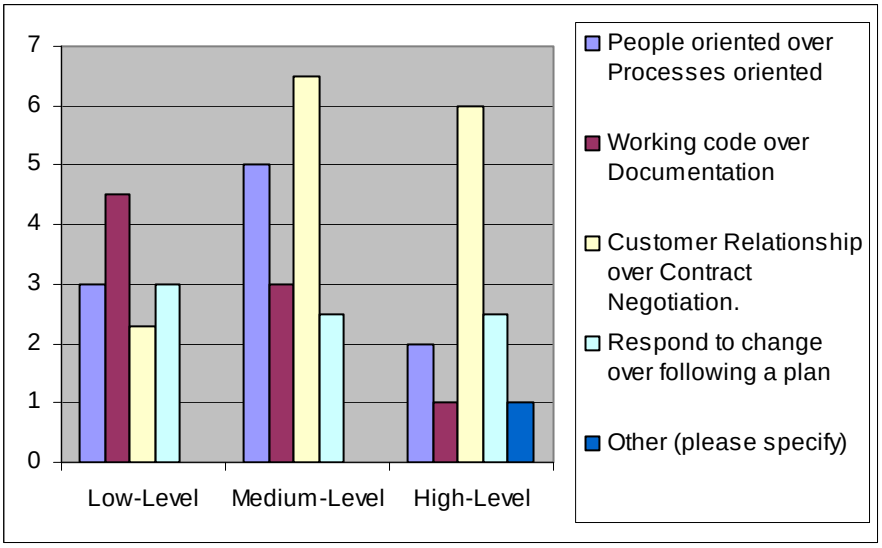
7. Which Agile Methodology do you mostly use for different kinds of Software development? *Please specify if more than one*



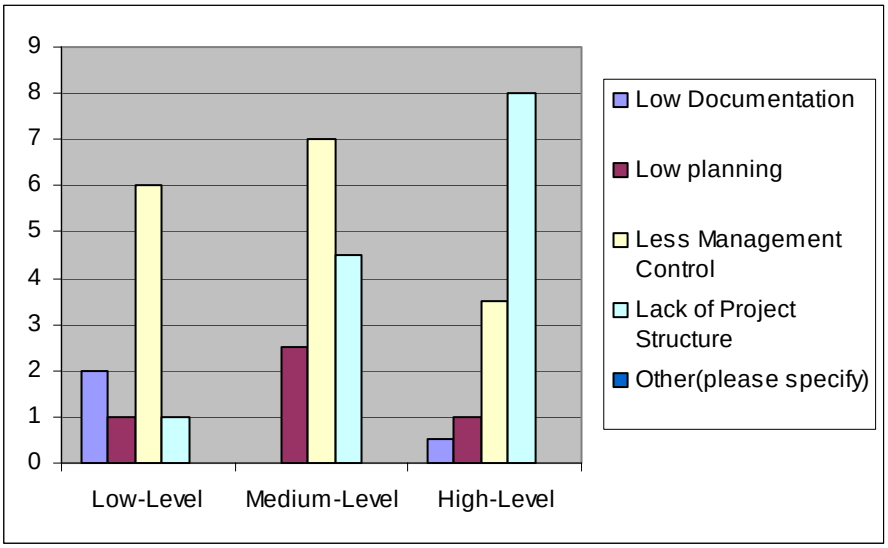
8. Which Heavy methodology do you mostly use for different kinds of Software development?



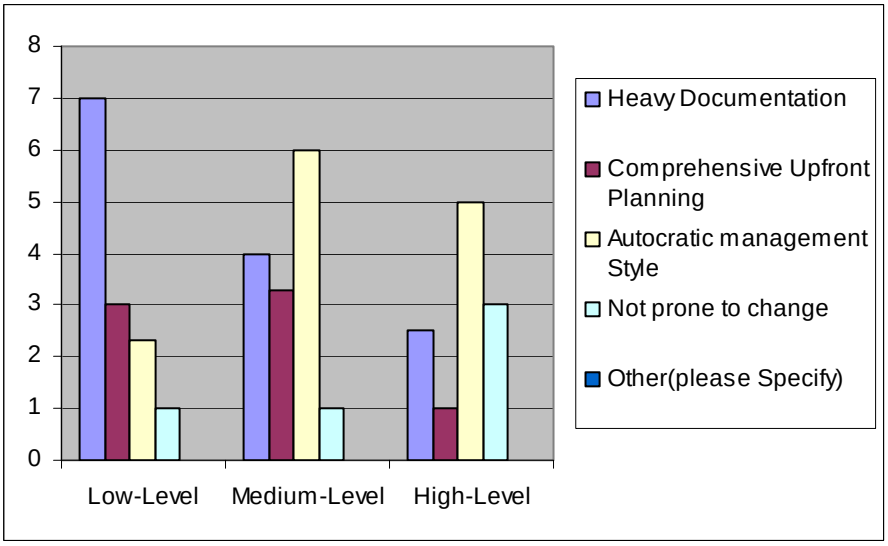
9. Which of the listed aspects of Agile Methodologies most appeal to you compared with Heavyweight Methodologies, for the 3 sizes of software development project?



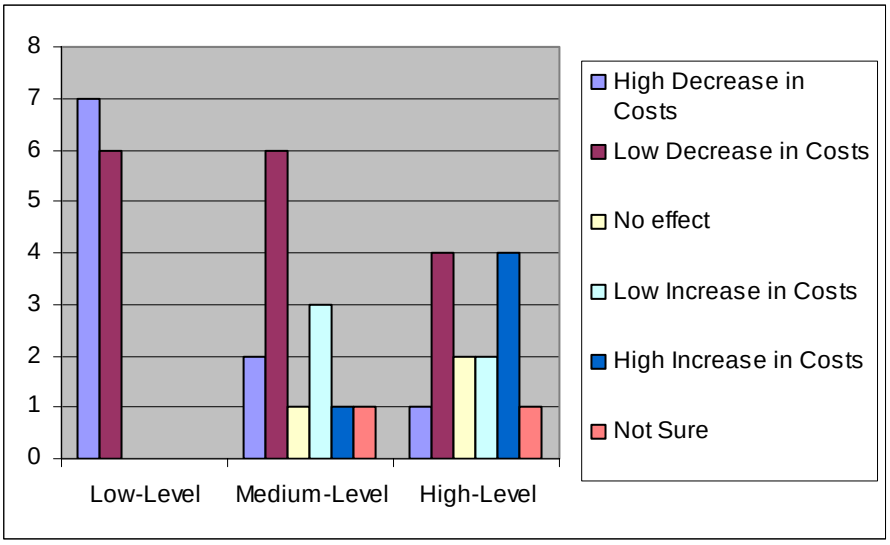
10. Which aspect of agile methodologies, do you dislike the most for different kinds of software development?



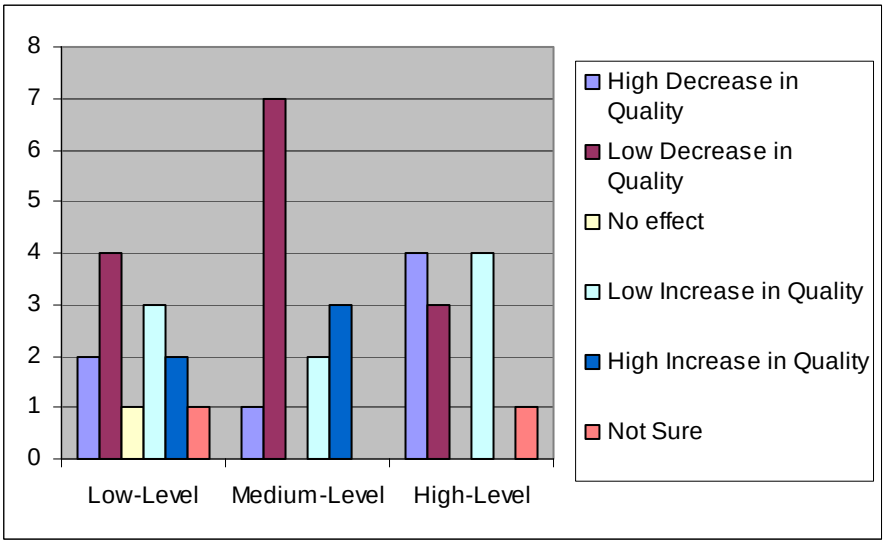
11. Which aspect of Heavy methodologies, do you dislike the most for different kinds of software development?



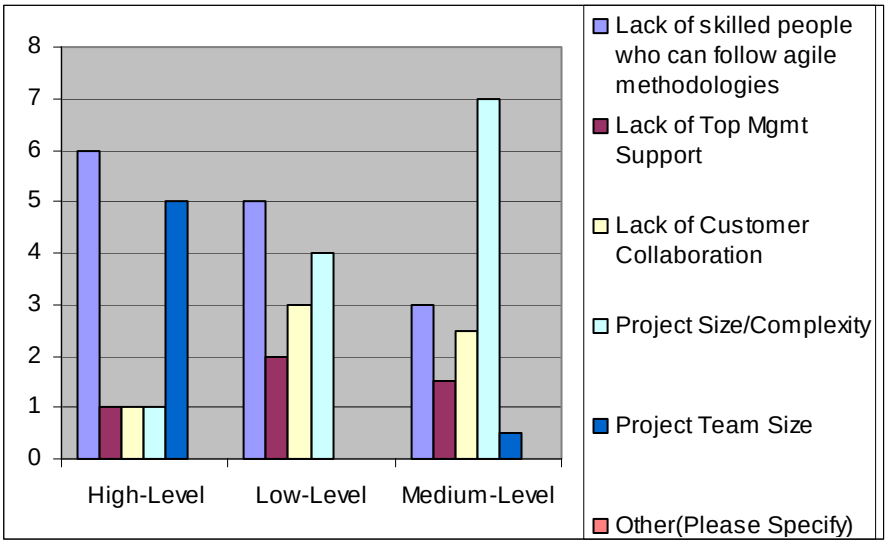
12. How do you believe that the cost of employing Agile Methodologies compares with Heavyweight Methodologies for the 3 sizes of software development project? *(select one in each category)*



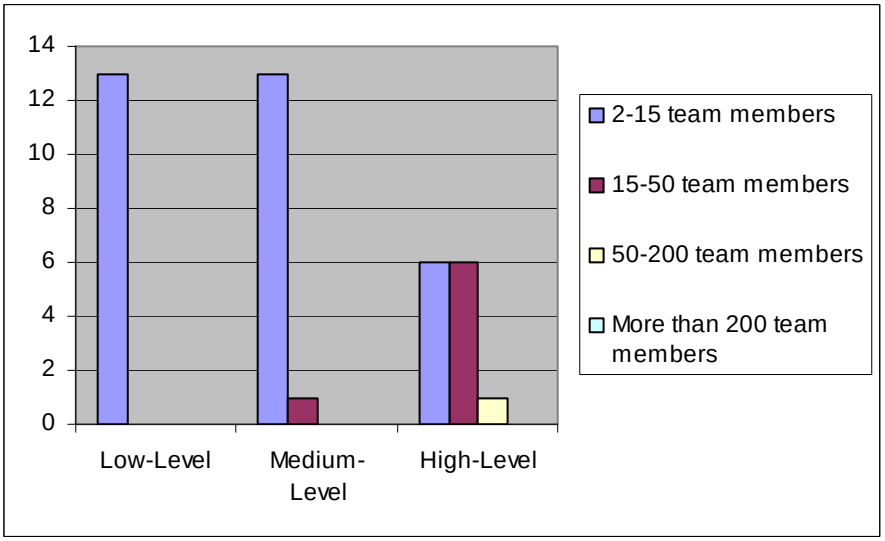
13. Do you believe that taking on of agile methodologies rather than Heavyweight methodologies have any effect on Software Quality for different levels of development?



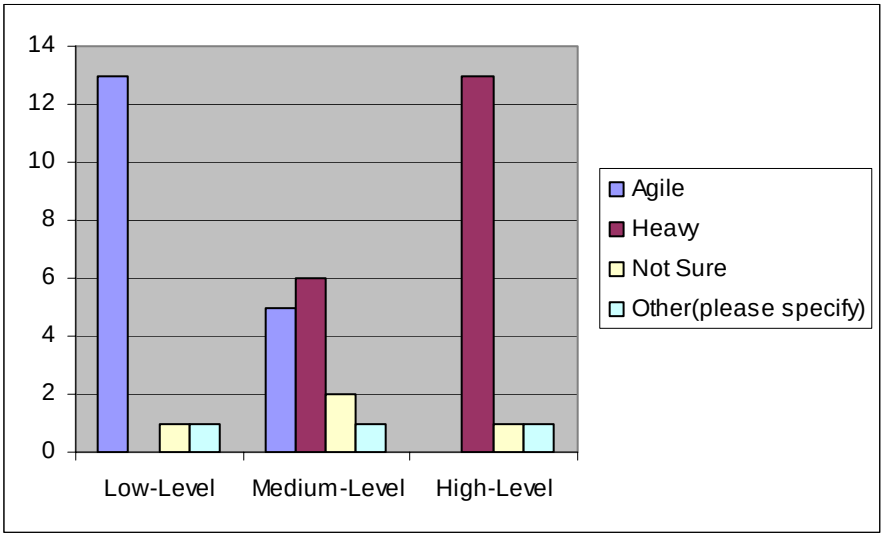
14. What do you believe is the most common problem experienced while practicing agile methodologies for different kinds of software development?



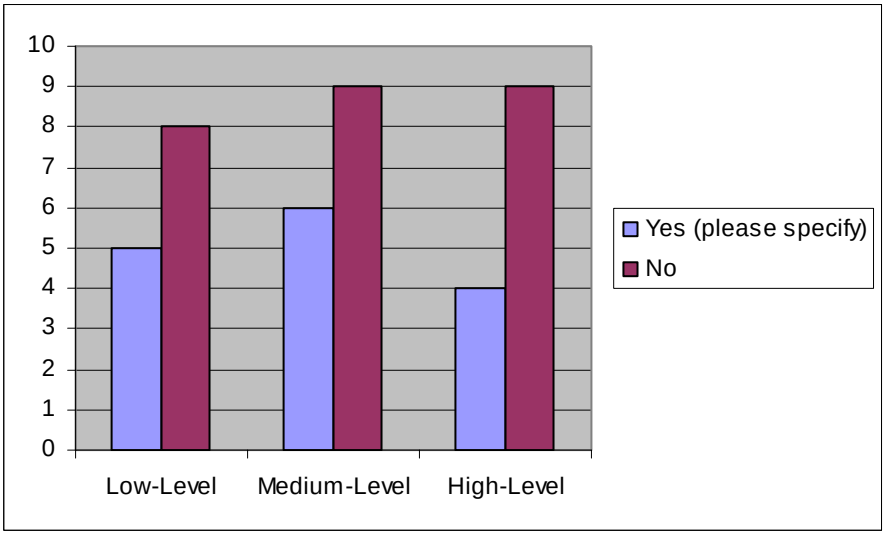
15. What is the Average Size of teams that work on Software Development in each project category, in your organization?



16. What do you believe is the most suitable methodology for the different kinds of Software Development?



17. Do you use any other methodology other than agile and heavy methodologies for different kinds of software development?



18. To what extent, do you follow different kinds of agile techniques for different kinds of software development?