Supporting Team performance - An empirical study of software teams, processes and tools to enhance software development

By

Shibani Basava

B.S., University of Colorado, Boulder, 2005

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirement for the degree of

Master of Science

Department of Computer Science

2008

This thesis entitled:
Supporting Team performance - An empirical study of software teams, processes and tools to enhance software development
has been approved for the Department of Computer Science

____
Kenneth Anderson

____
Tamara Sumner

____
Dennis Heimbingner

Date _______________

The final copy of this thesis has been examined by the signatories, and we
Find that both the content and the form meet acceptable presentation standards
Of scholarly work in the above mentioned discipline.

HRC protocol # 1107.18 – 01

Basava, Shibani Raj (M.S., Computer Science)

Supporting Team performance - An empirical study of software teams, processes and tools to enhance software development.

Thesis directed by Associate Professor Kenneth Anderson

In the software industry, companies have been known to employ ad-hoc processes or organizational standards to select a software development life cycle. The selection of an inappropriate life cycle can result in a project that does not satisfy customer needs, is delivered late or has exceeded the initial budget assigned to the project. In order to mitigate the risks in making an inappropriate selection much research has been conducted since the 1980s to quantitatively and systematically determine which life cycle and/or software tools are appropriate for a particular project. Studies in process tailoring have produced frameworks that aid in adjusting or tailoring a currently used life cycle to a particular project's goals and environment. Although several process selection models have been proposed as a result of these studies, most are designed for a particular point of view—project characteristics, risk factors, requirements stability, delivery schedules, etc. In this thesis, we propose a Unified Process Selection Model (UPSM) that encompasses many of the viewpoints considered in past research and applies them to modern software life cycles.

The purpose of UPSM is to assist project managers in selecting an appropriate software life cycle by looking at the project from different angles before making a decision. Empirical data was collected from a range of real project teams working in a variety of application domains (embedded, desktop, systems, tools) through interviews to gain an understanding of how these projects and teams make use of their selected software life cycle. This data has been used to evaluate the effectiveness of the UPSM. This thesis identifies areas where a currently selected software life cycle can be combined with another to provide benefit to a project and also identifies software tools that could assist in bridging the different life cycles. The purpose of the UPSM discussed in this thesis is to provide software project managers with a unified approach to selecting an appropriate life cycle that best suits a given project.

Dedication

For my parents, Venugopala and Veeraveni, who have supported and encouraged me throughout the course of this thesis and my life.

Acknowledgements

I would like to thank my advisor, Professor Kenneth Anderson, for his time and guidance throughout the scope of this thesis. Without his contributions and commitment to guiding the course of my thesis, this document could not have been written. I am forever grateful to the time and energy he has donated towards the successful completion of this thesis. I am also grateful to the participants of the study who contributed their time and experiences to assist this research. Thank you.

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

One of the major challenges that project managers face when developing a "successful" software product is the selection of an appropriate software development life cycle. As a result of increased demands for quality, productivity and consistency from the software industry in the past three decades, a need for disciplined principles to manage software development, operation and maintenance emerged. In answer to the demands, several development methodologies were created ranging from the conventional Waterfall [Royce, 1987] to the latest Agile techniques [Manifesto, 2001]. However, the criteria for selecting an appropriate development process for any given project are still unclear. Companies have been known to employ ad-hoc and informal methods to choose a life cycle for a project without clearly considering all possible implications of the process on the project's goals, characteristics and risks. A "wrong" process can result in a product that does not satisfy customer needs, suffers from longer development times, involves duplication of work effort, high costs, reduced quality and, high risk [Jurison, 1999]. Selection of an appropriate software development process can contribute to alleviating these issues.

1.1 Inappropriate project life cycles

An "inappropriate" life cycle can be defined as a software development model that is inept at meeting a project's characteristics, does not conform to the project's environment and diverges from the project's goals. Valuable time and money is lost if, for example, a Waterfall methodology is applied to a project with vague and unstable requirements. Geographically separated teams lose the advantages of quick face to face communication which is characteristic of Agile techniques [Manifesto, 2001]. Building a prototype when the user needs are well understood and stable would be an improper use of a project's development time [Alexander & Davis, 1991]. Selecting an appropriate development life cycle—that agrees with a project's characteristics and works towards a project's goals, rather than against them—can significantly increase the quality and productivity of the software being produced while decreasing the associated risks.

1.2 Related Work

Since the 1980s, much research has been conducted in developing quantifiable and systematic means for software life cycle selection and conforming the process to a project's goals and environments. Numerous models were proposed for evaluating processes using a particular viewpoint to aid in life cycle selection. [Basili, 1985] proposes a quantifiable model for evaluating software methodology by decomposing project goals into measurable metrics. [Basili & Rombach, 1987] presents an improvement methodology that iteratively evaluates the process, methods and tools against a project's goals and environment, and continually evolves with future projects. [Davis, Bersoff & Comer, 1988] introduces graphical comparison for assessing alternate software models with respect to user needs and cost. [Alexander & Davis, 1991] provides a set of twenty criteria to apply to the project and different processes to find the best suited model for the project. [Kettunen & Laanti, 2004] evaluates the various processes with respect to a project's problems and risk factors for embedded systems. Although, these models are effective when applying them to their specified viewpoint, they fail to consider the project from a global perspective.

In order to attend to this issue a Unified Process Selection Model (UPSM) is proposed to assess software process models with respect to multiple viewpoints.

1.3 Research Questions

The purpose of this thesis is to suggest a process selection model that evaluates modern software development practices from different viewpoints and proposes a lifecycle that is best suited for the project. The main question that was addressed in this thesis was: “*How do you select a software development lifecycle that best suits the project needs?”*

In order to propose a suitable development lifecycle, it is important to tailor it to the project needs. This research also addresses the following sub-questions:

- *“How do you tailor the selected lifecycle to the project’s goals?”*
- *“What tools would you use to assist in the tailoring and development process?”*

1.4 Research Approach

A qualitative approach was applied to this problem in the form of interviews with representatives from a range of real project teams in various software domains (systems, embedded, desktop, tools) to understand the relationship between projects, processes, teams and tools. The collected empirical data was used to profile the various development life cycles and practices used in the current software industry and to develop a process selection model that addresses the questions stated in Section 1.3. The UPSM performs the evaluation and selection of a software life cycle by looking at the project from different angles.

The organization of this thesis is as follows: Chapter 2 describes the context and background to process improvement and tool integration in industrial settings. It also discusses the motivation behind the work performed in this thesis. Chapter 3 highlights the related work done in the field and how they affect this study. Chapter 4 is a detailed description of the approach taken to perform this study and its results. Chapter 5 details the UPSM and performs a comparative analysis against the data collected from the field. Chapter 6 covers recommendations for future work in this area and Chapter 7 concludes this thesis with a discussion of its contributions.

CHAPTER 2
Context and Background

[Jalote, 2002] provides a useful analogy for understanding the importance of the topic of this thesis: “[Deciding what process to use] is a critical issue because much of the engineering activity will be governed by this decision. It is like going on a long driving trip—the planned route determines the course you will drive [p35].” Using Jalote’s trip analogy, choosing an incorrect route for the trip can increase the time it would take to reach the destination. Longer trips would be expensive with respect to fuel and tax the traveler’s patience. Furthermore, unfavorable weather, road constructions and unpleasant traffic conditions are all things that can be encountered and would be additionally discouraging. Similarly, in software development, selecting an inappropriate process can lead to “longer trips”, higher costs and higher levels of frustration among the project’s stakeholders. Project characteristics, analogous to road and weather conditions, can heavily affect the project’s development course. This chapter introduces the rationale about the importance behind selecting the “right” life cycle for a given project and presents background related to process and tool adoption.

2.1 Selecting the “Right” Software Development Life Cycle for a Project

Brown et al. introduce a project management AntiPattern called “Lifecycle Malpractice” in [Brown, McCormack & Thomas, 2000]. An AntiPattern is a description of a commonly recognized solution to a software development need that “generates significantly negative consequences” within the organization or the project

[Brown, McCormack & Thomas, 2000]. The Lifecycle Malpractice AntiPattern occurs when management fails to pick an appropriate software development process for a project. Brown et al. describe the symptoms and consequences of the Lifecycle Malpractice AntiPattern as follows:

> “The primary symptom of implementing an inappropriate lifecycle is the lack of knowledge about the status of the development. The main consequences are unexpected costs and software delivery delays. Without knowing the status of the development, the managers cannot properly manage in the information vacuum and any decisions are entirely subjective and risk laden. When a risk or problem occurs, the lifecycle phase process does not always deal with it in a timely and effective manner. This leads to spending more effort, with associated cost and delay, in resolving the problem [Brown, McCormack & Thomas, 2000: p291].”

Selecting the “right” development life cycle can mitigate the negative impacts on the progress of the project and the end product. The main advantages of following the right software development process are:

1. The cost of the project is within the assigned budget.
2. The product is released on time and in some cases earlier than was expected.
3. The final product satisfies customer needs.
4. Integration of customer or market-driven changes in the requirements is managed.
5. Unnecessary activities and excessive documentation is avoided for projects that do not need them.
6. Chances of risks being introduced decreases with controlled development phases.

7. Progress of the project can be seen clearly and predictions can be made regarding the direction of the project.

Brown et al. attribute the occurrence of the Lifecycle Malpractice AntiPattern to the ignorance of management in understanding the business and technical aspects of a life cycle. An example of this AntiPattern can be seen in a failed software project demonstrated by [Ewusi-Mensah, 2003] – "Highmark's HighBAR Project".

Highmark, a billion-dollar health insurance company, contracted the services of KPMG consulting group in May 1999 to develop an electronic billing and accounts receivable system for the company at the cost of 15 million dollars. KPMG estimated the release of the final product by the end of first quarter in 2002. The project was divided into four phases, each phase succeeding the previous one. Unfortunately, the project was shutdown and abandoned after only 20% completion of the first phase. KPMG requested an amount of 8 million dollars in addition to the initial 12 million for the completion of just the first phase. Highmark refused the request and KPMG had to abandon the product because of cost overrun.

The failure of HighBAR can be accredited to a number of organizational and managerial reasons. The project owners failed to satisfy the needs of their customer and to provide regular reports on the status of the project. The requirements were poorly understood and coding was allowed to start while the design of the project was still in progress. As a result of that decision, changes to the design caused code to be refactored or entirely redone. Further changes to the requirements as a result of poor understanding caused changes in the successive stages. Consequently, the project exceeded its cost margin and rework on the code and design pushed the schedule out further than anticipated. The software development process employed by KPMG failed to handle changes to the requirements and subsequently changes to the design and code [Ewusi-Mensah, 2003].

Selection of a different life cycle could have potentially prevented such a dramatic “end” of the software. As the requirements of the project were poorly understood and were bound to change in the future, partial implementation of the functionality should have been considered. Interactions between the project owners and the customers were non-existent and attributed to further disappointment for the client. Involving customers during the requirements process for every implementation of the partial functionality would improve the quality of the product and would have promoted the relationship between the consumer and the producer. Quick and incremental releases of major functionality would satisfy the customer needs and lighten the burden on the project’s schedule. Prototyping of less known features and performing demos with the customer would assist in understanding the requirements of the project. The selection of a development life cycle for the project was poorly made without consideration for the project’s needs and the needs of the customers. An incremental and iterative software development model with rapid prototyping elements would have been more appropriate for a project with such characteristics.

An exception to the Lifecycle Malpractice AntiPattern that [Brown, McCormack & Thomas, 2000] recognize is related to an organization’s level of maturity with respect to the Software Engineering Institute’s Capability Maturity Model (CMM). For companies considered immature or at Level 1 (Initial) of the CMM, choosing an appropriate life cycle will make little difference in achieving repeatable success across projects. Brown et al. propose that a proper process could be selected by the nature of the projects they are applicable to: “Demos and proof-of-concept prototyping”, “Incremental new technology application delivery”, and “Stable application maintenance”. For example, future changes to a project in its maintenance phase can be facilitated using the Waterfall process model as the requirements are familiar. However, for new project development, while requirements are still being discovered, an iterative or spiral model would best suit a project’s needs. Brown et al.’s strategy for solving the Lifecycle Malpractice AntiPattern is to identify a suitable process and develop a “super life cycle” by adapting it with different life cycles.

2.2 Successful Process Selection Case Studies

There have been several projects over time that have benefited from the selection of a different, more appropriate life cycle. Two such projects are the Singapore banking project [Highsmith, 2002] which switched to Feature Driven Development and the Chrysler Comprehensive Compensation project [C3 Team, 1998] which implemented Extreme Programming.

2.2.1 Singapore Banking Project

The Singapore banking project was a highly complex, commercial lending application that ambitiously tried to apply a document-driven approach during its first implementation. A large, well-known systems integration firm worked on the project for two years, releasing 3,500 pages of use cases, an object model with hundreds of classes, thousands of attributes as deliverables but no code. The project failed miserably in the first cut due to the large scope and multiple features that were poorly handled through use-cases. Jeff De Luca and Peter Coad, the creators of Feature Driven Development (FDD) pooled their ideas together and implemented this project as the first FDD project. A month was spent working on the overall object model, followed by a week of feature decomposition and planning. A proof-of-concept was developed soon after to exhibit the feasibility of the process and the project to the client. Within 15 months and with 50 employees, 2,000 features were delivered and the project was completed on time, under budget and to the client’s satisfaction [Highsmith, 2002]. This project demonstrated the improvement in the project’s quality, productivity and rate of success when an appropriate process was utilized for development. The main reasons for the failure of the first approach was the unnecessary time spent in developing all the specifications for the project without actually doing any coding. De Luca and Coad’s FDD focused more on the project’s features and delivered the software per feature set allowing the project to finish in a timely fashion.

2.2.2 Chrysler Comprehensive Compensation Project

The Chrysler Comprehensive Compensation Project (C3) involved the creation of an employee payroll management system. The project was initially implemented using the traditional Waterfall methodology and failed. The process was found to be too cumbersome and complex. It was redone using Kent Beck’s Extreme Programming approach. Instead of requirements specs and design documents, story cards and whiteboards were used. The project was released in iterations with heavy customer interaction throughout the life cycle of the project. Instead of planning for the future and generalizing the application as is usually recommended, focus was given to simplicity and only the things that were needed were implemented and later refactored in the released code. Scope, quality, resources and time were used as measurable values to communicate the project status to the customers and to higher management [C3 Team, 1998]. The project was very successful with the new methodology having eliminated the heavy waterfall ideals and focusing on what was needed. Table 1 shows the comparison between C3's old approach and the adopted Extreme Programming approach.

Table 1: Comparison between C3's Old and Extreme approach [C3 Team, 1998: p28]

OLD WAY	EXTREME WAY
Limited Customer Contact	Dedicated Customers on Team
No metaphor	Good metaphor
Central up-front design	Open evolving design
Build for the future	Evolve to the future, just in time
Complex implementation	Radical simplicity
Developers in isolation	Pair programming
Tasks assigned	Tasks self-chosen
Infrequent big-bang integration	Continuous integration
GUI-driven	Model driven
Fear	Aggressiveness
Ad-hoc workspace testing	Unit/Functional Testing
Limited top-down communication	Communicate, communicate, communicate

2.3 Tailoring the Software Process for a Project

Another area of risk that project management experiences is an organization's tendency to adhere to a single process model as noted by Brown et al.'s "One Size Fits All" AntiPattern [Brown, McCormack & Thomas, 2000]. Brown et al. discuss management's predilection to apply a single development life cycle across all projects in order to maintain consistency within the organization. The initial life cycle selected is refactored and added on to make it compatible with all of the organization's projects. This works well if the company is small and has few projects with similar characteristics. However, as companies grow large, different project types are developed and the same process model will not work for every one of them. Brown et al. recommend "tailoring" the process to a project's characteristics.

Tailoring can be defined as "the process of adjusting a previously defined process of an organization to obtain a process that is suitable for the particular business or technical needs of a project [Jalote, 2002: p38]". Often a chosen life cycle for the project does not completely satisfy the project's needs and goals. Tailoring the process by adding, removing or modifying activities can obtain a life cycle that is suited for the particular project. The management at Infosys, a globally successful information technology industry assessed at Level 5 (Optimizing) of CMM [Jalote, 2002], acknowledges that selection and tailoring of an appropriate life cycle is critical as it directs much of the project activity. Infosys applies a standard development process across its projects after tailoring it to a project's needs. A variation of the traditional waterfall process, Infosys' development process is broken into smaller phases that can be executed simultaneously with others. For example, system testing is divided into "System test planning" and "System testing". The "System test planning" phase can be executed in parallel with the "Detailed design" phase of the system, and the deliverables from both would serve as entry criteria for the "System testing" phase of the project. Figure 1 depicts the development life cycle employed by Infosys projects.

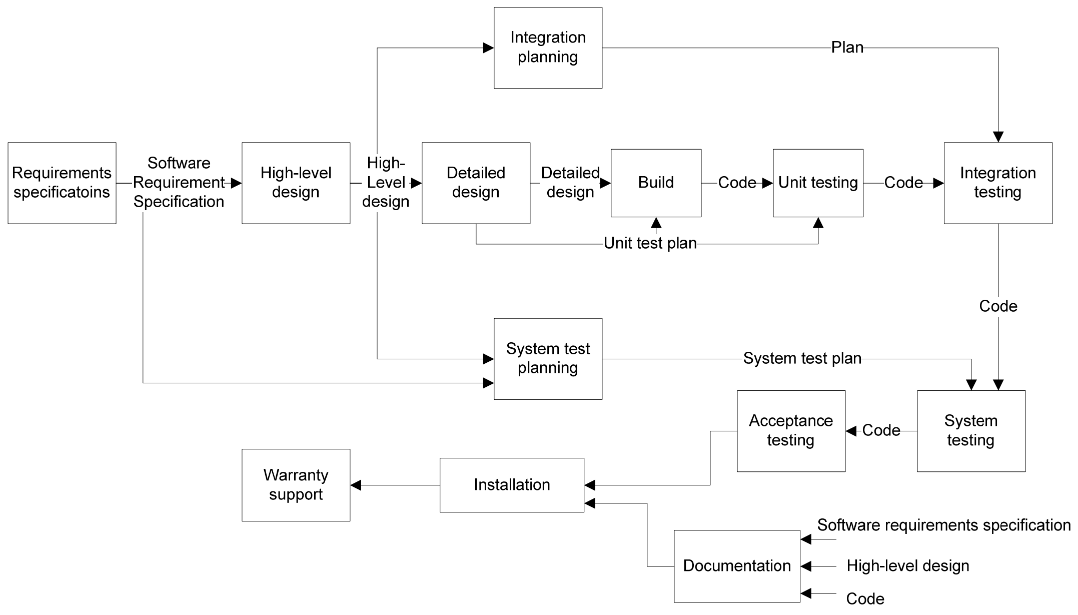


Figure 1: Infosys development process

Infosys' project managers are advised to make necessary adjustment to this process to suit their projects needs. Tailoring guidelines are provided to the managers at two levels: "Summary-Level" and "Detailed Tailoring". At summary-level, the project managers identify the characteristics of the project, prioritize them, and then apply general rules to the project activities based on the characteristics. The detailed-level tailoring covers the execution of the project activities, review and documentation needs.

Application of the same process to all of a company's projects can result in an uncontrolled growth of the process activities in order to support all possible project needs [Brown, McCormack & Thomas, 2000]. Tasks have to be continuously tacked on to the standard process in order to support future developments directly, increasing the planning time and cost of the project. Another consequence of having a standard process is that managers and developers within the company may end up creating their own versions of the development life cycle under the heading of the corporate process. Yahoo!'s attempt at controlling their software development process through a company wide application of the Waterfall Process called "Product Development Process" in 2002 is an example of this behavior [Sutherland & Schwaber, 2007].

The Product Development Process initiated by Yahoo! management was rolled out globally in 2002 and was made a mandatory process for all projects. As a result, a lot of teams ignored the process or made it look like it was being implemented. The teams that did implement the process suffered slow development time and heavy overhead that comes with the Waterfall process. They found little benefit in implementing the process. Although the management felt in control of the projects, in reality the teams rebelled until efforts were taken to apply Agile techniques. Tailoring a development life cycle to the project's needs can avoid such issues and promote a more flexible and creative environment for development.

2.4 Analysis of Software Development Life Cycles

When software engineering was still in its early years, development teams employed the "Code and Fix" development process [Schach, 2007]. "Code and Fix" allowed teams to develop an initial system with bare knowledge of the requirements and then refine the system iteratively until a feasible product was created. As the software industry was still fresh, this process worked well; however, as companies grew larger and spanned multiple domains from assistive software to airplane control systems, "Code and Fix" was no longer an option for development. It did not provide a stable code base. Continual iteration of system development without a plan or clear understanding of the requirements resulted in very unmanageable code. As a result, more refined development practices were proposed with the Waterfall process

[Royce, 1987] being one of them. Since then many development processes have appeared to address various issues experienced with the conventional methodologies. This research considers a subset of the modern development methodologies in order to narrow the scope of the study as well as to focus on the framework for selection rather than the actual development methodologies. The following five development processes are evaluated and used to demonstrate the Unified Process Selection Model: Waterfall [Royce, 1987], Spiral Model [Boehm, 1988], Scrum [Sutherland & Schwaber, 2007], Extreme Programming [Beck, 1999] and Feature Driven Development [Palmer & Felsing, 2002].

2.4.1 The Waterfall

The Waterfall life cycle model was developed to manage large software systems by Dr. Winston Royce [Royce, 1987]. The Waterfall process is a document-driven approach to software development where the process is sectioned into management phases that produce concrete deliverables in the form of documents. Figure 2 shows the Waterfall process and the output deliverables generated at the end of each phase in the life cycle.

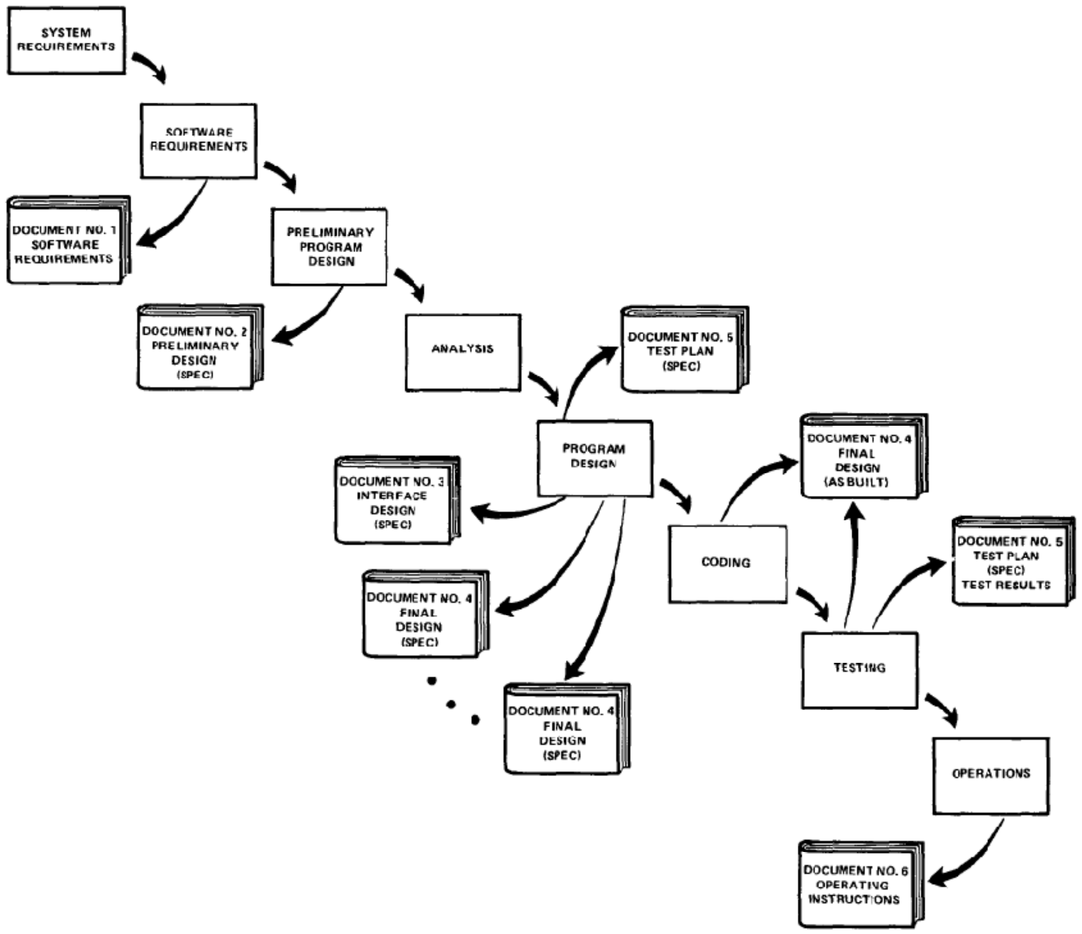

SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
ANALYSIS
DOCUMENT NO. 5 TEST PLAN (SPEC)
PROGRAM DESIGN
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
CODING
DOCUMENT NO. 4 FINAL DESIGN (AS BUILT)
TESTING
DOCUMENT NO. 5 TEST PLAN (SPEC) TEST RESULTS
OPERATIONS
DOCUMENT NO. 6 OPERATING INSTRUCTIONS

Figure 2: Waterfall process along with output deliverables at each phase [Royce, 1987: p333]

As seen in the figure, the Waterfall process starts with defining the system and software requirements. The software requirements specifications document is generated at the end of this phase. The requirements specifications are used as input in the preliminary design of the system to identify storage, timing and data-flux constraints in the system [Royce, 1987]. A preliminary design specification is created and fed to the analysis phase. Test plans are created from the requirements analysis and requirements specifications. Once the analysis is complete, the project design starts. Detailed documentation from interface specifications to full design specifications are created to provide tangible means of communicating the design of the system to a project's stakeholders. Royce argues that with good documentation, management can focus personnel on fixing defects in various sections of the code; without documentation, the responsibility lies solely with the creator. Coding follows the final design phases of the system and implements the documented design interfaces and specifications. Testing is performed on the completed code using the test plan generated after the requirements analysis phase. The operational phase of the system is accompanied by an operations manual to be used by the system's users.

While the Waterfall process is described as a linear execution of phases, several versions of the process exist that address problems with the original formulation. For projects where quality and integration of the code is given preference, testing is performed in parallel with code implementation. For new project development a pilot program of the actual system is created to understand the problem better. The following table describes the characteristics of the Waterfall process.

Table 2: Characteristics of the Waterfall process

Advantages	Disadvantages	Applicability	Potential areas for tools insertion
The system is well documented. Each phase in the development life cycle corresponds to the project management phases. Waterfall methodology is a disciplined approach to software development. During staff	Software is usually delayed because of excessive documentation. Low customer interaction and high change frequency throughout the development process leads to software that does not satisfy user needs. Requirements need to be fully understood	Requirements are known and stable. Users know what they want and can express the needs articulately. Heavy documentation is required. Project risks are relatively low. The project is sufficiently large. Small projects do not	Tools to create, manage, and organize requirements. Should provide traceability to design and test plans. Tools to facilitate system integration and

turnover, or when the chief personnel leave the team, information is persisted in the form of documents that can be used to bring new comers up to speed. Initial costs and schedule estimates are lower and more accurate when the requirements are known and stable. Waterfall method works well for projects in maintenance as requirements are understood and the developer is familiar with the problem.	before project begins. As users are the primary source of requirements accessible only at the start of the project, they need to know what they want. Harder to respond to and implement frequent changes to the requirements. New tools and technologies make implementation strategies unpredictable [Sutherland & Schwaber, 2007] Mistakes made early in the process are often not found until much later; thereby increasing the cost and time to fix the delays [Palmer & Felsing, 2002]. Heavy documentation is excessive for small and simple projects. A working product is not available until late in the life cycle [GSAM, 2003]. Cost of fixing defects increases the later they are found in the development life cycle. As more changes are	scale well as documentation is excessive.	automated test runs. Tools to manage phase completion and visibly track the progress of development through document deliverables. Tools to assist in the design process (e.g. UML tools)

	integrated, the gap between specifications and code widens.	

2.4.2 Spiral Model

Contrary to the Waterfall methodology's document-driven approach, Barry Boehm designed the Spiral development model as a risk-driven approach [Boehm, 1988]. The classic spiral process workflow is shown in Figure 3. The radial dimension represents the cumulative costs incurred to date. The angular dimension represents the progress through the project life cycle. Each cycle of the spiral corresponds to a phase in the development.

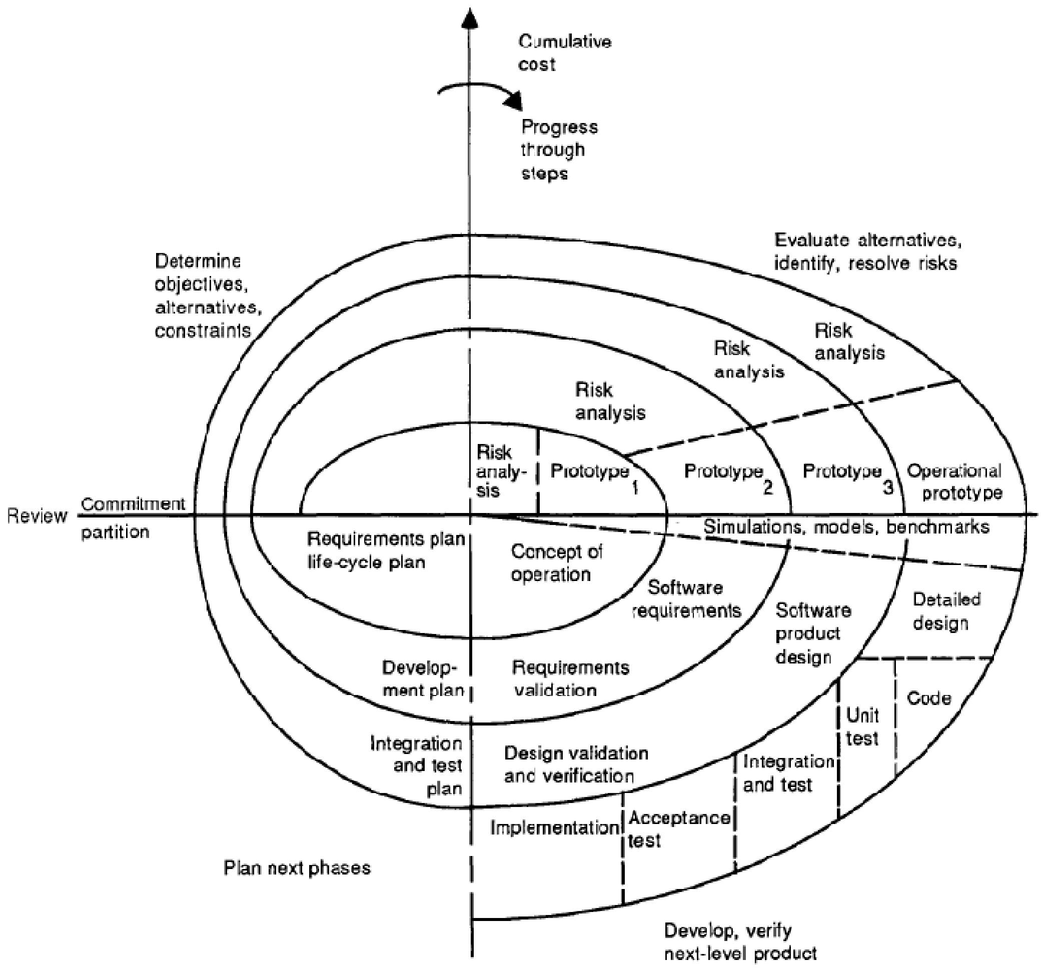


Figure 3: Spiral Model of software process [Boehm, 1988:p64]

Each cycle begins by identifying objectives for the current phase, alternative means to address the objectives, and constraints imposed on the application of the alternatives. This process leads to the development of a strategy to address the identified objectives. The strategy is then analyzed for risks and solutions are investigated, sometimes by creating proof-of-concept prototypes to attempt at mitigating the potential risks in the system. The proof-of-concept prototypes provide scale models for feasibility analysis of a solution and are used to discover additional risk factors in the system. A project may be terminated early if certain risks cannot be mitigated (e.g. release of software with same functionality by a competitor). If all risks are successfully mitigated, the next step in the phase follows the basic waterfall approach. The process can be modified to incorporate incremental development when necessary. Each level of specification is followed by evaluation of the phase and development of plans for the next phase in the spiral [Schach, 2007 & Boehm, 1988].

Table 3: Characteristics of Spiral Model

Advantages	Disadvantages	Application	Potential areas for tool insertion
Spiral model supports reuse as alternatives are explored earlier in the development. Post-delivery maintenance is another phase in the spiral; not very different from development [Schach, 2007]. Continual risk analysis provides information of areas that cannot be	Spiral model is complex and hard to manage [GSAM, 2003]. Development costs and time are usually greater [GSAM 2003]. If the immitigable risks are encountered at later stages in development, terminating the software is considered a breach	User needs are very important [GSAM, 2003]. Project has high risks areas. Requirements need to be refined before design [GSAM, 2003]. Problem is complex and relatively large in size. Development needs to be responsive to	Quick prototyping tools are beneficial to evaluate solutions for mitigating risks. Tools to track risks and estimate their impact on the project costs and delivery schedules. The tool should be able to generate risk analysis reports.

mitigated, allowing projects to be terminated early in order not to incur costs of developing risky software. Software quality objectives are incorporated into the product development earlier [Boehm, 1988]. Spiral modeling provides a viable framework for integrated hardware-software development. The principles used for software development can be applied to hardware construction as well [Boehm, 1988]. Spiral development models the behavior of other development practices when risk analysis provides the right conditions. Prototypes provide scaled models for feasibility analysis.	of contract with external customers [Schach, 2007]. Continual risk analysis increases the cost of development. If the project is small, the cost of risk analysis may equal the cost of the whole project development. Spiral model drastically increases the cost of small, simple projects. Skilled risk analysts are required; else the analysis is of no value when risk areas escape detection. Assumes that software is developed in discrete phases.	user needs and unsteady requirements by analyzing uncertainties. Problem is critical. Managers are skilled at risk analysis. Project is regulated by a government body like the FDA. User is flexible to project delays.	Tools to integrate the system while using an incremental and iterative approach to development within spirals. Tools to track requirements and trace them to design and test cases. Having the capability to assign and track their completion within iterations is beneficial.

2.4.3 Scrum

The Scrum framework of software development established by Ken Schwaber and Dr. Jeff Sutherland is an extension of the incremental and iterative process. An Agile process, the Scrum process is designed to address frequent changes in environment, internal and external requirements and changes to schedule. Figure 4 depicts the Scrum workflow for a project.

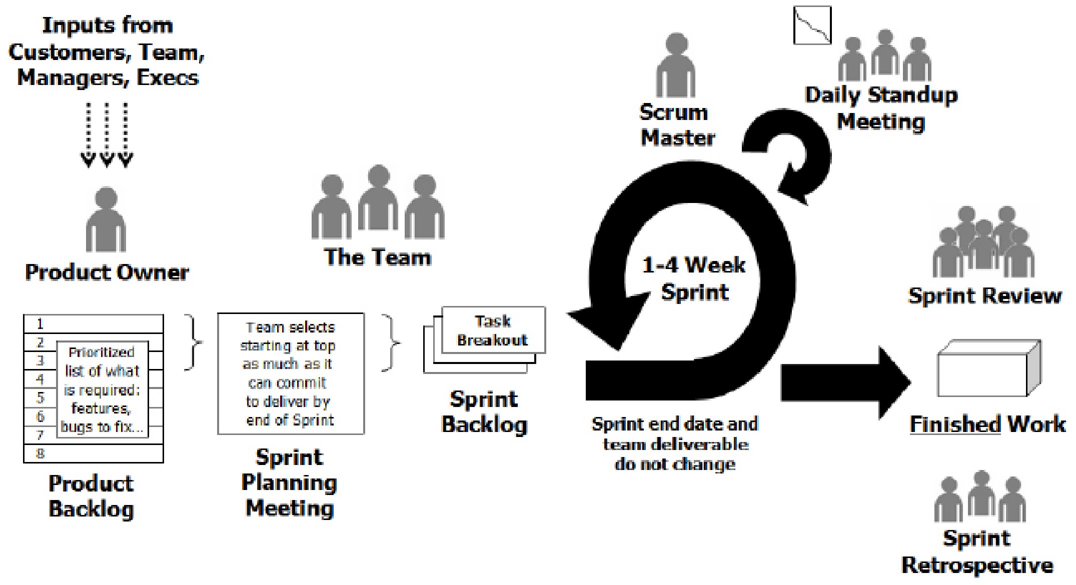

Inputs from Customers, Team, Managers, Execs
Product Owner
The Team
Scrum Master
Daily Standup Meeting
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Prioritized list of what is required: features, bugs to fix...
Team selects starting at top as much as it can commit to deliver by end of Sprint
Task Breakout
Sprint Backlog
Finished Work
Product Backlog
Sprint Planning Meeting
Sprint end date and team deliverable do not change
Sprint Retrospective

Figure 4: Scrum development process [Sutherland & Schwaber, 2007: p22]

The Scrum workflow is comprised of three main phases: Pre-Sprint (the planning phase), Sprint (the implementation phase), and Post-Sprint (the project closure phase). As can be seen in Figure 4, the development process starts with a Product Backlog where a prioritized list of all project requirements, defects and environmental issues is established by the Product Owner using inputs from customers, managers, teams and executives. The Product Backlog is an “active” list that can be edited and maintained by the Product Owner throughout the development period. During the Pre-Sprint, a sprint planning meeting is called where the goals of the upcoming sprint are established by going through the Product Backlog and establishing the amount of work hours to be spent for each task in the list. The team selects the tasks they want to implement and estimates the time it would take to implement. The tasks and estimates are committed to the Sprint in a fixed Sprint

Backlog which cannot be changed for the duration of the Sprint. Each Sprint is a fixed iteration generally spanning 1-4 weeks. The end date for the Sprint is maintained regardless of whether all the tasks are completed. Daily Scrum meetings (less than 30 minutes) are held to discuss the tasks completed since the last meeting, the tasks to be completed before the next meeting and obstacles in the process. Towards the end of the iteration a demo is performed for the users to gain feedback and to suggest improvements or changes for the next Sprint (Sprint Review and Sprint Retrospective). Each iteration ends with tested shippable code. Releases are roughly decided in a Release Backlog which is a subset of the Product Backlog and can be modified at any time during the process. The progress is assessed using a Burndown graph that measures the work hours remaining compared to the estimated work hours for each Sprint [Highsmith, 2002; Sutherland & Schwaber, 2007].

Scrum is one of the most applied Agile practices present. Yahoo!, Microsoft, Google, Lockheed Martin, Motorola, SAP, Cisco and Capitol One are few of the successful implementers of the Scrum project development process [Sutherland & Schwaber, 2007]. The following table details the advantages, disadvantages, application and potential tools of Scrum.

Table 4: Characteristics of Scrum

Advantages	Disadvantages	Application	Potential areas for Tool insertion
Team decides how much work will be completed in one Sprint instead of being assigned to it. Documentation is kept light stating only	New critical features cannot be introduced in the middle of a Sprint. The process does not maintain heavy documentation often	Projects that require early code deliveries. Daily communication can be scheduled. User requirements are not clear and are	Organize and share Product Backlog (open to suggestions from everyone but

what is necessary. Requirements exist as a live list of prioritized customer needs and can be modified throughout the project life cycle. Features cannot be randomly added to the current Sprint making each iteration a finite list of tasks to be executed. Regular status reports of the tasks completed and tasks in progress are presented in the form of Daily Scrum Stand-up meetings. Sprint Burndown chart provides visible indication of the project progress. Sprint end date and deliverables do not change. Visible indication of things that are not working well early on in the project. Works with distributed teams and large teams by cutting teams into separate groups. Goals are clear for each Sprint.	required by regulatory bodies. Design is performed along with coding and testing during each implementation. This may cause heavy refactoring to be done during each iteration in order to accommodate requirements that do not work well with the previous design. The process can get chaotic as new requirements are entered into the Backlog continuously and design keeps changing to accommodate it.	established using demos and prototypes. Developer does not fully understand the requirements. Requirements are unstable and change frequently. Flexibility is required in release schedules and functionality to be released. The team sizes are small (or can be cut into separate groups) Heavy customer involvement is required (in the form of users, stakeholders, Product Owners) Visible progress is required in the form of functionality.	only editable by the Product Owner) Tools to use the Product Backlog to create and organize subsets as Sprint backlogs and Release backlogs Automatic generation of progress reports (Burndown charts) Continuous integration within each Sprint prevents long hours to be spent integrating the whole project during the final Sprint before the release. Tools to manage project schedule by tracking hours spent on each Sprint. Should be accessible to

			all members of the team working on the Sprint so that updates can be made regarding the status of the tasks.

2.4.4 Extreme Programming

Extreme Programming (XP) was developed by Kent Beck as a flexible evolutionary design approach to software development focusing mainly on customer satisfaction [Succi & Marchesi, 2001]. The principles of XP stem from the need to address volatile requirements and to produce simple designs. Like all Agile practices, XP is a set of guidelines and principles which when employed produce certain expected results like simple design, increased feedback, quality and communication. XP can be described by the rules and principles it advocates [Beck, 1999]:

Planning Game – Customers decide the release and scope of delivery based on cost and time estimates provided by developers. Stories are created and assigned to an iteration until the final release. Only key requirements are implemented every iteration.

Small releases - The project is divided into short iterations (daily and monthly). Product is put into implementation in a few months and code is released frequently.

Metaphor – The shape of the system is determined with a set of metaphors to focus the design and implementation.

Simple design – System is designed based on what is needed at present and not what will be needed in the future. This is done to avoid making the design overly complex and resistant to changes in the future.

Tests - Automated unit test cases are developed before coding begins. This allows tests to be executed as soon as the code is written to provide instant feedback. In addition to unit testing, customers develop a set of functional acceptance tests to be run against the system. When defects are found, new test cases are written to test the defect.

Refactoring - Continuous refactoring is performed on the code to make it adaptable to changes in future iterations. The existing code is replaced and optimized internally without changing the behavior of the application.

Pair programming - Code is written by a pair of programmers working on the same terminal: one writes the code, the other reviews it.

Continuous integration - The system is continuously integrated (every few hours) and tested using tools to provide instant feedback as soon as code is written.

Collective ownership - The code is owned collectively by the team, allowing any member of the team to change any piece of the code at any time. For this purpose coding standards are maintained.

On-site customer - Throughout the project development life cycle heavy on-site customer interaction is expected.

40-hours week – Overtime is avoided. A 40-hour week philosophy is employed in order to avoid getting into situations where too much effort is being placed to get the work done.

Table 5: Characteristics of Extreme Programming

Advantages	Disadvantages	Application	Potential areas for tool insertion
Customer selects what needs to go into each release and what is the highest priority. Cost and schedule estimations are done before each iteration. Each iteration produces tested and shippable code of new functionality. Pair programming implements robust code where defects are detected earlier through code inspections. Individual experience also increases. Iterations are kept small (< 3 weeks), so progress can be seen early in the development period. Design and process is flexible to integrate changes in requirements. Design is simple as it is only designed to address present requirements and not designed for future. Traceability is maintained between	Uses heavy refactoring which requires skill and experience. A novice developer may end up refactoring the code so much every iteration that the current iteration may get delayed. (This scenario is usually avoided by placing novice programmers with experienced ones in pair programming) Heavy customer interaction is an essential component of XP. If users are not easily available, XP suffers. If developers are not used to working in pairs, it can affect the productivity of the code, as they may squabble over the code. Collective ownership becomes a problem when one developer changes the code unexpectedly while another is using it. (Code changes need to be managed) Information about the software is present in code, stories, test cases	The team is small (< 10 people) Team members are co-located. Members are comfortable working in pairs. The requirements are unstable and change continuously. Users are easily accessible and can participate in the development process throughout the development period by taking control of the iterations and developing acceptance tests. Heavy documentation is excessive for the project. Partial functionality is demanded early in the life cycle. Users are not clear about what they want. Developers are not familiar with the problem domain, or the project is new to the industry or software in general.	Tools to create and organize stories. The stories should be accessible by customers and developers. The customers should be able to prioritize the stories and assign it to a release. Should be able to generate a release plan with the targeted stories. Should be able to create tasks in relation to the stories. Accessible by developers in order to be able to add estimations and pick which tasks they want to work on. Should be able to generate progress report on the iteration status based on tasks completed. Tools for continuous integration and testing of the software; flexible enough to add acceptance tests

requirements and the code (smaller gap between specification and code) [Succi & Marchesi, 2001]. Documentation is light, giving the developers chance to work more on the actual code. Evolutionary design allows developers to learn and optimize the code [Succi & Marchesi, 2001]. Quality of the software is usually better as tests are run and the system is integrated continuously.	and memory. If system is not changed for a long time or if someone leaves the company, the knowledge is limited to code and test cases and part of the information is lost as no documentation is persisted [Nawrocki et al., 2002]. (Pair programming usually assists in retaining the knowledge within the team) Oral communication is used for information exchange which is susceptible to forgetfulness when not written down [Nawrocki et al., 2002].	Users are not able to communicate their requirements without first looking at demos. Defects are to be addressed during the iteration and implemented for the coming release. Early indication is required when the project seems to be failing. The system is new and relatively small or medium sized [Beck, 1999]. Requirements are vague [Beck, 1999].	developed by user. Tools should provide traceability between stories, test and release. Defect management system that tracks the defect, the test case associated with the defect and the releases it was found and fixed in. The continuous integration system should be able to generate failure reports and productivity reports based on the tests passed and the tasks left to complete from the iteration release plan. Capability to add information to the stories in the form of design and specifications would assist in persisting information in case of team turnover.

2.4.5 Feature Driven Development

During the development of the Singapore Banking Project in 1997-98, Jeff De Luca and Peter Coad pooled their ideas together to apply a Feature Driven Development approach to the project that had already been declared “undoable” once before (Section 2.2.1). Feature Driven Development (FDD) provides a client-centric, scalable solution for complex projects. Figure 5 depicts the FDD process model.

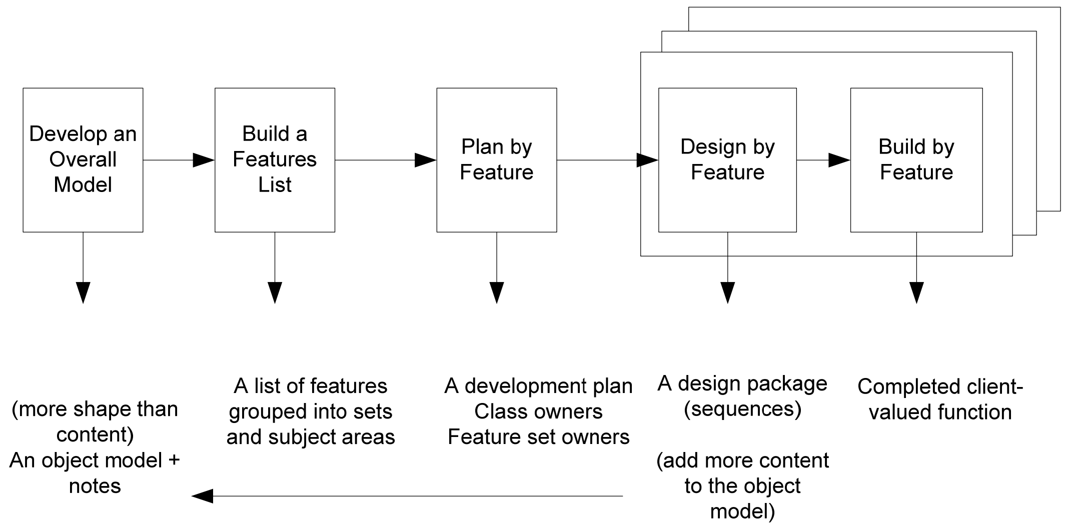

Develop an Overall Model
Build a Features List
Plan by Feature
Design by Feature
Build by Feature
(more shape than content) An object model + notes
A list of features grouped into sets and subject areas
A development plan Class owners Feature set owners
A design package (sequences)
(add more content to the object model)
Completed client-valued function

Figure 5: The five processes of FDD with their outputs [Palmer & Felsing, 2002: p58]

FDD process is flexible enough to be incorporated into other processes; however there are steps described by the FDD approach which when executed together, provides a mechanism to derive repeated results from the process. These steps as shown in Figure 5 are described below [Palmer & Felsing, 2002]:

Develop an Overall Model – Chief Architects (experienced object modelers) work together with domain experts (customers, stakeholders) to perform high-level walkthrough of the system followed by detailed domain walkthroughs to determine the scope of the modeling. Domain models are generated in groups consisting of domain experts and developers and a proposed model is selected to represent the domain in a consensus. The domain model is merged into the overall model and the shape is adjusted. This process is repeated iteratively until an acceptable overall object model is developed for the development. This process usually takes around 10% of the development effort. This model is updated throughout development.

Build a Features List – Using the domains identified in the first step, the system is decomposed into subject areas (major feature sets), which is further decomposed into business activities (feature sets), and furthermore into features (the smallest client-valued step within a business activity). Features are represented using the <action><result><object> template and take no more than two weeks maximum to implement.

Plan by Feature – A high level document plan is developed by the Program Managers, Development Managers and Chief Programmers. Business activities or subject areas are sequenced into iterations based on dependencies and priorities. The iterations are time boxed into formal releases for large complex systems. Class owners are identified and assigned to key classes in the system.

Design by Feature – A Chief Programmer is assigned to develop a number of features. The Chief Programmer selects a set of features from his list to implement and forms appropriate feature teams using Owners of the participant classes. The teams develop detailed sequence diagrams for the feature and the Chief Programmer refines the object model. Inspections are held to validate the design. This step is performed per feature.

Build by Feature – Code is implemented using the design packages (sequence diagrams) developed in the previous step. The code is unit-tested and reviewed once completed and then promoted to the build by the Chief Programmer if successful. This step is a per-feature step and produces client-valued features that have been code-inspected and unit-tested.

Table 6: Characteristics of Feature Driven Development

Advantages	Disadvantages	Application	Potential areas for tool insertion
Project complexity is handled well by decomposing the problem into manageable features for implementation. Features are small (can be completed in under 2 weeks) and measurable such that clients can see the progress on a frequent basis [Palmer & Felsing, 2002]. FDD, unlike some of the other Agile practices, is scalable for teams larger than 10 people in size. Front-end design reduces the refactoring effort during the implementation phase. Design is persisted in class and sequence diagrams and information is not lost when a member leaves the team. Project failure can be determined earlier	Class ownership can cause developer of one class to wait on changes to be made in another, causing delays to pile up the more this happens, slowing the pace of development [Highsmith, 2002]. With class ownership, when the class owner leaves the project it takes some time to bring someone new up to speed, especially for significant classes [Highsmith, 2002]. Features must be known and prioritized. Once the features have been selected, it is very hard to change the contents without causing serious damage to the project [Kettunen & Laanti, 2004]. In large complex projects with too many interdependencies, it is difficult to isolate and sequence feature	Project is large and complex. Partial functionality is required earlier during the development. Users are knowledgeable about the domain. Users are willing to participate in the domain object modeling. Developers are skilled and experienced in object modeling. Users are able to express the needs of the system in terms of when certain features are required. The requirements are vague and can be clarified through domain walkthroughs. Changes are regular but do not exceed the 10% limit of the initial feature list.	Tools to maintain and track Class Owners and which feature teams they currently belong to, Feature Owners and which features they are currently working on; accessible to all members in order to find out who to talk to for implementing changes to a class, or allow Feature Owners to determine who to pick for Feature teams and their availability. Tools to create and maintain a prioritized list of feature lists, with their hierarchical business feature decomposition. The tool should be able to generate reports based on the features complete and

as coding is done earlier. Customers take part in deciding the priorities of the features to implement. The level of documentation can be adjusted and described by the Chief Programmers. Feature design inspections and feature code inspection milestones root out potential errors for each feature. Communication among members increases as new feature teams are created per feature implementation.	sets. If only business features are considered, there is risk of ignoring internal technical features [Kettunen & Laanti, 2004]. For embedded systems, it might be challenging to develop feature lists with concurrent hardware development [Kettunen & Laanti, 2004]. Developers need to be very skilled in order to develop good object models. FDD addresses only the software construction process. The requirements gathering and system tests are beyond the scope of FDD [Kettunen & Laanti, 2004].	When the new feature list contains more than 10% of the master feature list, then changes to the schedules and scope of the project are made by the management [Palmer & Felsing, 2002]. Chaotic and frequent changes in the requirements increase the cost, the time and produce fewer features than planned for.	should be able to assign feature sets or subject areas to a release. The reports can be evaluated against this date. [Palmer & Felsing, 2002] describe six milestones in the Design by Feature and Build by Feature phase. Tools can be used to assist in management of these milestones and generating completion reports on these milestones. Change management tools are helpful to track new requirements separately from the main features list and apply the 10% rule against it as described by [Palmer & Felsing, 2002]. Tools that map feature to objects provide traceability between design and requirements. Automatic

			generation of feature lists relieve manual documentation overhead.

2.5 Using Tools to Assist in Software Engineering

"Much of the rapid, even frantic advances in all aspects of computer hardware was brought about by productivity-enhancing tools such as computer aided design (CAD) and computer aided manufacturing (CAM) tools [Simon, 1993: p4]." In response to this sentiment, software engineers employed the use of software tools to assist in development of projects since the early 1980s. Computer Aided Software Engineering (CASE) tools evolved from the need to automate software development activities and to propose a new approach to the software life cycle concept which features increased use of integrated tools to increase overall productivity and the quality of the generated software [Mcclure, 1989]. [Brown et al., 1994] define CASE tools as follows:

> "A CASE tool is a computer-based product aimed at supporting one or more software engineering activities within a software development process [p14]."

During the development of software, a number of activities are performed ranging from creating cost and resource estimates to writing the user manual. Although, most activities cannot be fully automated, software tools can be used to assist in the execution of these activities. Upper-CASE tools are a class of CASE tools that assist in the front-end activities of the development life cycle like requirements analysis, specifications management, requirements traceability etc. RequisitePro, SharePoint and Microsoft Project are examples of tools that fall in this category. Lower-CASE tools are back-end tools that assist in the implementation workflow, code management, system integration, automated testing, defect management and post delivery maintenance phases of the development process. Tools that fall in this category are usually compilers, editors, configuration management tools like CMVC, source control tools like CVS, Subversion and Visual SourceSafe, and defect tracking mechanisms like BugZilla and FogBugz.

Selecting the right software development process involves identifying areas of improvement in the methods selected. It is important to tailor the development process to address the needs of the environment and project in order to successfully apply the process. One way to cope with the overhead of implementing the improvements is to introduce tools to address the needs. However selection of tools is not a straightforward process, especially as there are many commercial tools available and there is also the possibility of implementing custom tools tailored for a specific organization's or project's life cycle. [Prather, 1993] discusses five common failure points in the evaluation and selection of CASE tools: (1) understanding the criteria needed to select a project, the function the tools need to serve, the scenarios a tool is applicable to and the environment the tool needs to be integrated with; (2) establishing the pre-requisites for the tools like the purpose of the tool, what kind of project is it addressing, a common vision shared by the members of the team, how flexibly the tool will support the development process etc.; (3) knowing the organizational characteristics, the maturity level of the organization, the level of discipline in their planning phases; (4) understanding the performance of the tool in the application scenarios and under stress; and (5) managing expectations and imagination of the team as high expectations are the prime factor in the failure of tools that do not meet the expectations. Failure in any one of these five failure points introduced by [Prather, 1993] can result in the ultimate failure of the software tool selected. However, failure of tools is not only attributed to their evaluation and selection but also to the areas where they impact the software process they are implemented in. [Bruckhaus, 1994] explains:

> "CASE tools have been installed with major process improvements in mind and have often failed to meet these expectations. It is clear that paradigm shifts in the process cannot be achieved through merely acquiring and installing new tools [p1]".

When introducing new tools to an environment it is essential to consider certain key issues: the areas of improvement, proper training of future users, determining how the tool will be used, quantifying the expected benefits, monitoring the application of the tool, measuring the actual benefits and taking corrective actions when the tool does not meet the expected benefits [Bruckhaus, 1993]. Bruckhaus has devised and introduced a detailed methodology for tool insertion while reducing impacts to the software process called the Method for Planned Tool Insertion (MPTI) [Bruckhaus, 1993] or the Tool Insertion Method (TIM) [Bruckhaus, 1994]. There are certain basic steps involved in the implementation of TIM: (1) specify the purpose of the tool, introduce quantifiable goals to measure the success of the method, create a process model to insert the tool in and define key areas of improvement in the process model; (2) survey and select tools based on the key improvement areas identified in step 1 and the goals defined for the tools to achieve; (3) customize the tool based on the process model specified in step 1 to satisfy the key areas of improvement; (4) iteratively validate and measure the impact of the software tools against the process to continuously redefine and improve the model; the measurement goals defined in step 1 assist in the validation of the process model and the tool; (5) implement the re-designed solution into field and train the users in their usage; (6) evaluate the tool in the field using the measurement goals and document the results to be used for future improvements; (7) institutionalize the tool for a given organization’s needs by resolving ownership and standardizing the procedures for the tool implementation in a process.

Regardless for evaluating the tools or inserting them in a software process, it is vital to understand the purpose the tool is to serve and the benefits of its implementation in the process. Blindly selecting and inserting a tool without establishing goals negatively impacts the software process by introducing unexpected costs and time overhead. The strengths and weaknesses of the tools must be studied to design effective insertion methodologies and a continuous evaluation is required to determine the usability of the tools. Using CASE tools can improve the overall development process by increasing the productivity and improving the quality of the software in an advanced environment, however it can also increase the costs and delay the software if not properly implemented.

2.6 Summary

This chapter presented the motivations behind this thesis and an overview of the software processes considered (Waterfall, Spiral, Scrum, XP, and FDD). Selecting the “right” development life cycle increases the chance of success of a project as could be seen in the Singapore Banking project and Chrysler Comprehensive Compensation project. Tailoring is necessary in order to address a project’s goals. A life cycle on its own will not be able to satisfy all of a project’s needs. This chapter also introduced the complexities associated with inserting new tools into a project and presented solutions to resolve them. In the next chapter, we discuss the studies performed in this field and introduce some process selection models proposed in the past.

CHAPTER 3
Related Work

The idea of developing a quantifiable and systematic approach to selecting a software development life cycle is not new. As software industry developed over the last three decades, the need for formal methodologies for selecting life cycles and performing process tailoring emerged. This chapter reviews the research and work done in this field by Basili [1985], Basili and Rombach [1987], Davis, Bersoff and Comer [1988], Alexander and Davis [1991] and, Kettunen and Laanti [2004].

3.1 Quantitative Evaluation of Software Methodology

Victor Basili describes a framework for quantitative evaluation of software methods and tools in his publication [Basili, 1985]. Basili introduces a measurement and evaluation paradigm based on identifying goals that highlight the needs of an organization and generating quantifiable questions that can be answered through distribution of data or metrics. He reveals that major problems in software development are related to the management's lack of ability to identify the criteria to select methods and tools for project development and to evaluate their usability when applied to their product. In this technical paper he establishes the fact that any method and tool cannot be put into a working environment and be expected to work with the project. Identifying an integrated set of tools that work across the development life cycle would help mitigate the erroneous situations when dealing with such a transfer. He defines the selection of such an integrated set of tools as follows:

> "An integrated set of tools must by definition be an abstraction. These techniques must be engineered for a particular environment. In this sense, software engineering involves the application of an integrated set of techniques to a specific project, with its unique problems, constraints and

environment. This approach requires an understanding of the project and environment in which it is to be developed so that the right set of techniques can be (1) chosen from the integrated set and (2) refined for the environment [Basili, 1985, p2]."

In order to evaluate the set of integrated tools and methods for an environment, Basili introduces the idea of models and metrics as quantifiable means that assist in characterizing the process and product by abstracting and simplifying the environment. A model is a view of the process or product that helps in understanding some characteristic of the software development process. The means to quantify the characteristics observed within the models are termed as metrics. Basili highlights two kinds of metrics in his paper: objective metrics (absolute measures taken on the product or the process) and subjective metrics (a relative measure of some aspect of the feature usually garnered through a consensus of opinions.) These metrics can be used for determining cost, quality, characterizing, evaluation, prediction, estimation and motivation. Basili's paradigm for the evaluation of methods and tools uses these measures for quantifying data collection to reach a conclusion.

The measurement and evaluation paradigm for selecting methods and tools described in this paper is based on understanding what data needs to be collected, why it needs to be collected and how that gathered data is interpreted. The first step in the process is to establish goals that clarify the needs of an organization or project. These goals are vague ideas of what the organization is trying to achieve through such a process. Once these goals have been identified they are defined in quantifiable terms as questions. Answering these questions will satisfy the goal identified. The questions are answered through a series of data collection techniques and metrics that are defined against the project. The metrics can be used to answer more than one question as they can be interpreted differently for different models of the software. The next step in the process is to develop a mechanism for data collection, i.e. interviews, forms, electronically etc. The data is then validated for accuracy followed by an analysis of the data in the context of the questions and goals they are associated with.

Basili describes that an iterative approach to such a paradigm in a work environment that analyses the findings for future projects builds a knowledge base for not only current managers but for new employees as well. As there are a number of methods and tools available, Basili argues that this paradigm alone is not enough to accurately evaluate them for a project. He identifies several approaches to quantitatively evaluate methods and tools and experiments with each of them. These approaches are characterized as follows: blocked subject-project (more than one project with more than one team per project), replicated project (with one project and more than one team for the project), multi-project variation (more than one project but with one team per project) and single project (one project with one team).

The case study using blocked subject-project analysis revealed that a great amount of statistical information can be gained for evaluation but the projects studied are small module size projects and the results may not match to a project of a larger scale. The same drawbacks were encountered for replicated projects, which could be applied to projects slightly smaller than regular, but the number of replications due to the design of the study (same process replicated between different teams) provided soundly supported results. In a multi project variation, the controlled factors of a methodology were varied across several similar projects to establish a relationship between it and factors like productivity and quality. There were several drawbacks to this study mainly caused by the lack of a control group and the small variation size for method evaluation that might prevent accurate data collection. For a single project evaluation approach the confidence level is much lower because of its sole reliance on a single team and project.

Basili's paradigm for software methods and tools evaluation was introduced in 1985. The quantifiable approaches defined in the paper are soundly supported with experimentation. This article approaches the evaluation of projects by understanding the environment and refining the process to suit it. It characterizes the goals of projects into quantifiable questions and attempts to answer them with measurable metrics and data. When applied to a software process, this approach could potentially assist in selecting the appropriate development life cycle and tools that could satisfy the goals of the project. For example, if one of the goals of a project is to provide shorter, faster release cycles, software process evaluation metrics could be based on error reduction rate, feature implementation time, feature prioritization, need for documentation etc, that could help answer questions like what kind of feature prioritization would provide shorter iterations, etc. Although the article is nearly 20 years old, it is a valuable resource for process and tool selection evaluation.

3.2 Software Process Tailoring using Defect Profiles

[Basili & Rombach, 1987] presents an improvement methodology to assist projects in tailoring their software process to the goals and environment of the project (the software process mentioned here is the global software process and the methods and tools that support it.) [Basili & Rombach, 1987] argue that the major problem in a software project is “the lack of management’s ability to (1) find criteria for choosing the appropriate process (global process model and methods and tools supporting those models), (2) evaluating the goodness of the software process, and (3) improve it [p345].” Basili et al. propose a solution for these problems in a tailoring methodology that characterizes the goals, methods, tools and the process using defect profiles and uses these profiles for adjusting the methods and tools in areas that have a high defect occurrence rate. The authors state that using defect profiles is only one of the possible characterization approaches; however they believe that this approach guarantees that all factors that could possibly affect the outcome of the project are taken into consideration.

Basili et al. propose an improvement methodology that is cycled through each release of the software and in future projects using a similar environment and process. The improvement process has been structured into five steps: (1) characterize the environment and the approach being used for the project (includes characterizing methods, tools, project factors, problems and resources using a characterization scheme—in this paper Basili et al. use defect profiles), (2) set up improvement goals, quantify them using questions and determine data to be collected for successful project development and improvement over the past projects (the authors propose the use of evaluation methodology presented in [Basili, 1985] for this step), (3) choose methods and tools for the project based on the characteristics of the improvement goals (the methods and tools should be chosen in order to meet these characteristics and allow easy collection of the data identified in step 2 for evaluation), (4) perform the software development and maintenance while collecting the prescribed data, validating it and providing improvement feedback for the current project in real time (the data can be collected using forms, interviews and automatically through the chosen tools), and (5) analyze the data post mortem to evaluate the current practices, determine problems, record the findings in a corporate database and recommend new improvement goals for future projects. The improvement process is repeated with the next project development and so on extending the knowledge for future development efforts.

As stated earlier, Basili et al. recognize that improvement goals, methods, tools and processes can be characterized using any number of mechanisms; however for this paper the authors have used defect profiles. Figure 6 shows the scope of the tailoring process that Basili et al. discuss in this paper.

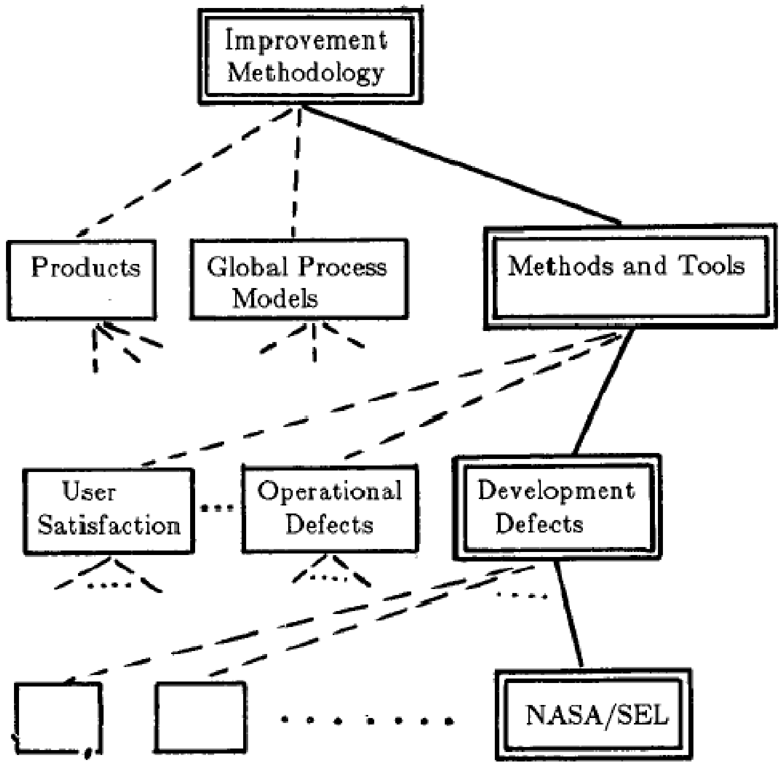


Figure 6: Improvement Methodology Framework [Basili & Rombach, 1987: p346]

[Basili & Rombach, 1987] identify three instances of defects used for characterization of the improvement goals – errors, faults, failures. Defects introduced by humans while trying to understand requirements, to solve problems and use methods and tools are called errors. Faults are defects within the software caused by errors. Failures are the departures of the project specification from the execution of the product. One error in the software can cause multiple faults and one fault can be caused by several errors. A failure in the system can be caused by multiple faults of the same and different kind. Errors can be prevented (through training etc). Faults can be prevented from entering a software product through the use of tools like syntax editors etc. During analysis of the project (code reviews, design reviews) faults can be detected, isolated and corrected. Execution and testing of the code can weed out the failures in the system and help isolate and correct related faults.

Basili et al. provide various schemes for classifying error, faults and failures to determine which methods and tools will help prevent them. The schemes presented in this paper have been evaluated using three criteria: can the defect class for each defect be determined, can the necessary information for decision be easily collected and for each defect class are there methods and tools that can prevent or detect, isolate and correct the defects. Errors can be classified using the time of their occurrence in the development life cycle (requirements, design code, unit test, system test, acceptance test and maintenance errors), and the domain which causes them (application, problem-solving, semantics, syntax, environment etc.). Faults can be classified by the phase or activity in which they are detected, whether they are caused by omission or commission and by the software aspects affected by the faults. The time of detection can be used to classify project failures (only those phases/activities related to execution are considered) along with the severity of the failure (show-stoppers, impacts production significantly, cosmetic changes etc.)

The tailoring process described by Basili et al. first establishes the improvement goals for the project using the methodology proposed in [Basili, 1985]. The environment is characterized by its impact on the product development and the quality of the resulting product. The methods and tools are characterized by what kind of defects they address or prevent. The defect schemes assist in understanding what areas of the project need improvement by analyzing the data generated by the current and past projects with similar goals and environment. [Basili & Rombach, 1987] apply the tailoring methodology to one of NASA’s ground support systems to demonstrate its usability and application of defect profiles. The authors also introduce TAME (Tailoring A Measurement Environment), a tool that supports the quantitative evaluation of the software process used in the project by providing features that assist with the different steps of the improvement methodology. TAME provides a data repository to store all collected data and a history to assist in future project development.

The tailoring process described in this paper provides a quantitative approach to characterize and evaluate methods and tools that are to be used for project development. The defect profiles introduced by Basili et al. is a feasible method for process evaluation as it helps detect areas where the applied software process is weak and can be improved for future implementations.

3.3 Comparison Model for Alternate Life Cycles

[Davis, Bersoff & Comer, 1988] aims at describing a paradigm to compare and contrast alternative life cycle methodologies, such as rapid prototyping, iterative development, evolutionary prototyping, and automated software synthesis against conventional methodologies like the waterfall process with respect to continually evolving user needs. Davis et al. advocate that the primary reason for schedule delays or the failure of software to meet user needs is that software is imprecisely aiming at a "moving target". The user needs are constantly evolving as shown in Figure 7.

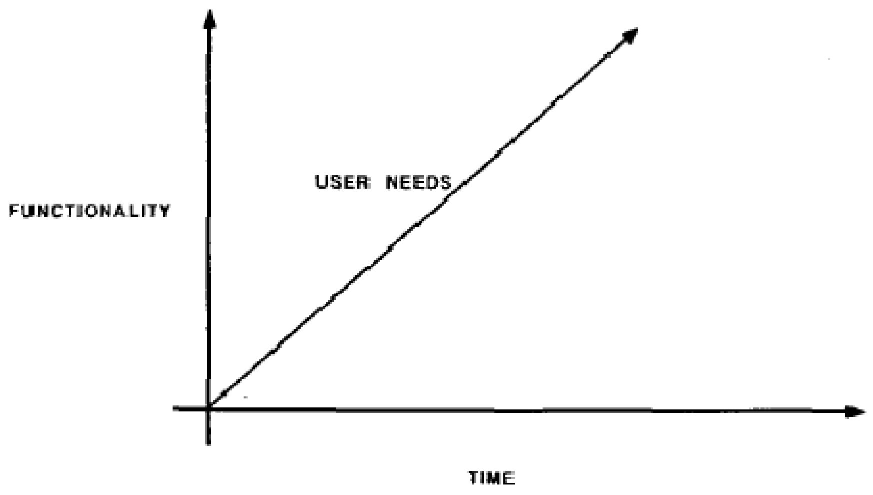


Figure 7: Constantly evolving user needs [Davis, Bersoff & Comer, 1988: p1455]

The figure represents the changing nature of user needs while recognizing that in reality the nature of the change is neither linear nor continuous. The X and Y axis represent the general measure of time and functionality respectively without constraining scale to uniform units.

[Davis, Bersoff & Comer, 1988] introduces four alternative life cycle methods and compares them with the conventional waterfall method using the following metrics: Shortfall (the measure of how far the software is at any given time, t, from meeting the current user needs), Lateness (the measure of how late the software is in meeting the new requirement in the market), Adaptability (the rate at which the software is able to adapt to the changing user needs), Longevity (the time from when a system is introduced and is changeable to when it is replaced by a new system), Inappropriateness (the measure of how much the software fails to address the other metrics). These metrics are graphically represented in Figure 8 for conventional processes. The shortfall of the software is the distance between the current system functionality and the current user needs. The lateness is the delay in time between when the new user functionality is introduced and when it is implemented in the software. The slope of the graph depicts the adaptability of the software life cycle and the area between the user needs and the conventional software life cycle represents the software's inappropriateness to meet the changing user needs.

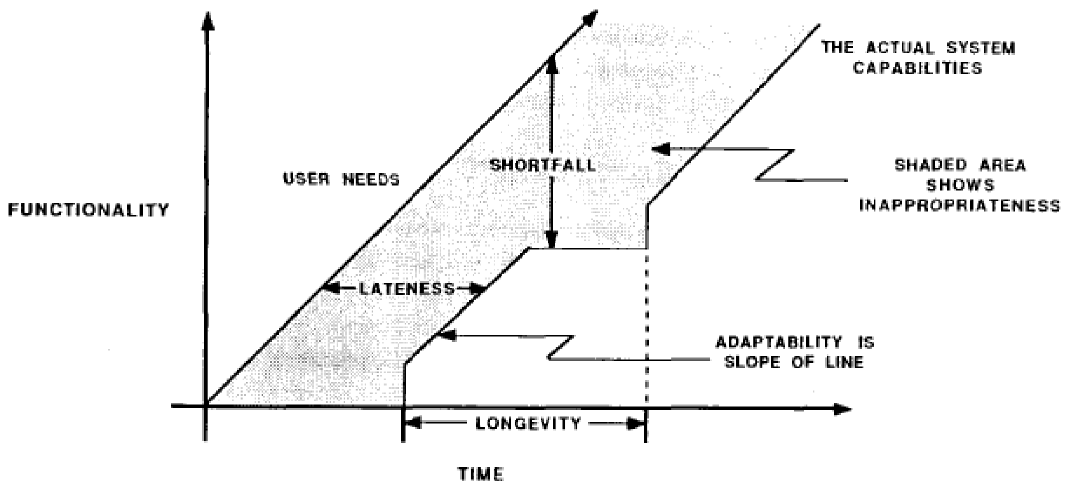

THE ACTUAL SYSTEM CAPABILITIES
SHORTFALL
USER NEEDS
SHADED AREA SHOWS INAPPROPRIATENESS
FUNCTIONALITY
LATENESS
ADAPTABILITY IS SLOPE OF LINE
LONGEVITY
TIME

Figure 8: Software productivity metrics [Davis, Bersoff & Comer, 1988: p1456]

[Davis, Bersoff & Comer, 1988] applies these metrics to both conventional and alternative software life cycle models (rapid prototyping, iterative development, evolutionary prototype and, automatic software synthesis.) Rapid prototyping allows a better understanding of user needs. Hence, the initial system introduced meets the previously identified market needs much better than a conventional life cycle model with only partial understanding of the requirements. Incremental software development leads to a step graph that has a steeper slope depicting a higher rate of adaptability than conventional methods. The steps are a result of discrete builds of the software that are built on top of the previous release. Evolutionary prototyping is considered a continuous development of prototypes with known requirements. The graph has a steep slope, almost meeting the current user requirements. The automatic software synthesis mechanisms that translate requirements to operational code have a very small area of inappropriateness due to its constant generation of an entirely new system with the current requirements. Davis et al. view the comparison paradigm as not only a method for providing comparison between different life cycles but also an insight into how to modify them to meet our situation.

[Davis, Bersoff & Comer, 1988] further explores the presented paradigm for software comparison to allow a methodology for software process selection. The article introduces the following aspects that affect the choice of the software process: requirements volatility (probability of change in requirements), the “shape” of the requirements volatility (gradual vs. discrete leaps), the longevity of the application and, the resources available for implementation. In addition to addressing the ever changing software functionality, a manager must make the appropriate selection to lower costs and development time, while meeting the defined selection aspects. A new dimension is added to the 2D software comparison graph keeping in mind the cost of software development. The 3rd dimension tracks the cost of the software development at any given time in the life cycle as shown in Figure 9.

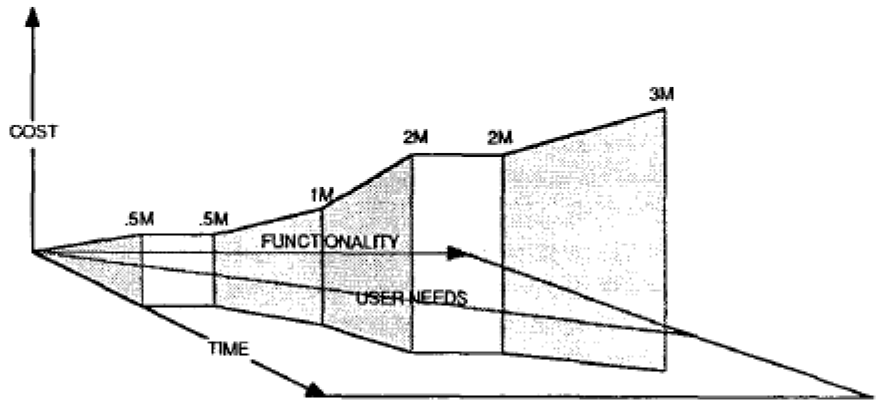


Figure 9: Possible cost analysis for conventional development [Davis, Bersoff & Comer, 1988: p1459]

The 3D comparison model provides insight into both functional and funding faults for a software life cycle and allows the selection with both factors in mind. The cost of making future upgrades to the software can be weighed against the cost of adding new functionality and the productivity can be measured in terms of the cost of labor and the functionality that is implemented per hour of the labor.

According to the paradigm presented in [Davis, Bersoff & Comer, 1988] an ideal software life cycle is one that continuously addresses the user needs at the lowest cost and the shortest time possible for development. The paradigm allows a comparison to be made between different software processes and aids a manager in selecting a methodology that meets the user needs while keeping the cost of development in mind.

3.4 Criteria based software process selection model

[Alexander & Davis, 1991] describes guidelines for selecting an appropriate process model based on a particular project. It defines a comprehensive set of criteria for evaluating different processes for a project and establishes a method to determine which process best suits a given project. [Alexander & Davis, 1991] argue that this method for selecting a software development process provides a higher rate of success than the current ad-hoc, unjustified means of selecting the process by a manager.

The methodology for determining an appropriate process uses a set of 20 criteria, C. Each criterion, c_i, is evaluated using an ordered vector of elements, V_i, with value, v_{ij} such as {novice, experienced, expert}. A binary indicator, s_{ij}, is used to identify which value of the criteria applies to a project. This value is project dependent and can only be evaluated based on the project characteristics. Therefore, the s_{ij} values form a characterizing matrix that describes the attributes of the project. The applicability of the process to the criteria values is depicted as a binary value, a_{ij}.

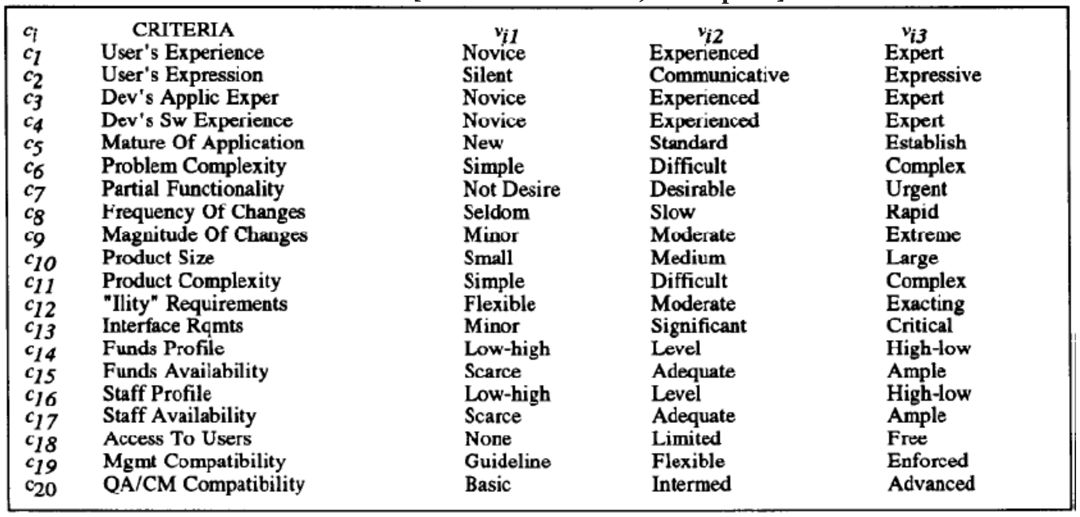
In order to determine the appropriate software development process model for a particular project, the project manager must first examine the project characteristics and determine the s_{ij} values for each criterion. Then, he must compute the applicability rating for each process model using the following formula:

$$RATING = \sum_{i=1}^{20} \sum_{j=1}^{3} (s_{ij} * a_{ij})$$

Figure 10: Process selection rating computation [Alexander & Davis, 1991: p522]

The process model with the highest rating is considered the best suited model for the particular project followed by the second in the ranking and so on. The set of 20 criteria that [Alexander & Davis, 1991] uses can be characterized as personnel (users and developers), problem (complexity, frequency of changes, partial vs. full functionality), product (size, GUI, complexity), resource (funding, staffing availability and application over the project time span), and organizational (compatibility to policies, configuration management) criteria. The criteria and their values are listed in the following table.

Table 7: Selection criteria and values [Alexander & Davis, 1991: p523]



c_i	CRITERIA	v_{i1}	v_{i2}	v_{i3}
c_1	User's Experience	Novice	Experienced	Expert
c_2	User's Expression	Silent	Communicative	Expressive
c_3	Dev's Applic Exper	Novice	Experienced	Expert
c_4	Dev's Sw Experience	Novice	Experienced	Expert
c_5	Mature Of Application	New	Standard	Establish
c_6	Problem Complexity	Simple	Difficult	Complex
c_7	Partial Functionality	Not Desire	Desirable	Urgent
c_8	Frequency Of Changes	Seldom	Slow	Rapid
c_9	Magnitude Of Changes	Minor	Moderate	Extreme
c_{10}	Product Size	Small	Medium	Large
c_{11}	Product Complexity	Simple	Difficult	Complex
c_{12}	"Ility" Requirements	Flexible	Moderate	Exacting
c_{13}	Interface Rqmts	Minor	Significant	Critical
c_{14}	Funds Profile	Low-high	Level	High-low
c_{15}	Funds Availability	Scarce	Adequate	Ample
c_{16}	Staff Profile	Low-high	Level	High-low
c_{17}	Staff Availability	Scarce	Adequate	Ample
c_{18}	Access To Users	None	Limited	Free
c_{19}	Mgmt Compatibility	Guideline	Flexible	Enforced
c_{20}	QA/CM Compatibility	Basic	Intermed	Advanced

The set of criteria described by [Alexander & Davis, 1991] is used to characterize the project attributes as well as determine the applicability of a process model to each value of the criteria. For example, for a waterfall process model, the frequency of changes (c_8) is not as often as perhaps in an agile process. Therefore, its applicability values for the criterion will be a_{81}= 1, a_{82} = 0, a_{83} = 0.

To facilitate the selection of the process models Alexander and Davis organized the process models by their level of abstraction into a three-level hierarchy as shown in the Figure 11:

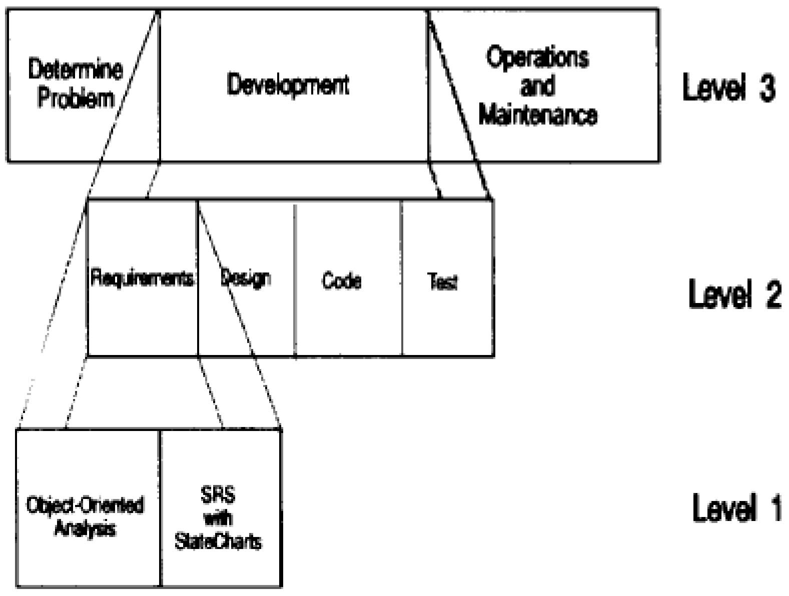


Figure 11: The Hierarchy of Software Process Models [Alexander & Davis, 1991: p524]

The top level of the hierarchy or level 3 contains process models that “delineate” groups of activities and assign them to certain time segments. They also address the partitioning of user needs like delivery cycles, partial vs. full functionality etc. In [Alexander & Davis, 1991], the authors highlight conventional, incremental and evolutionary models as level 3 process models.

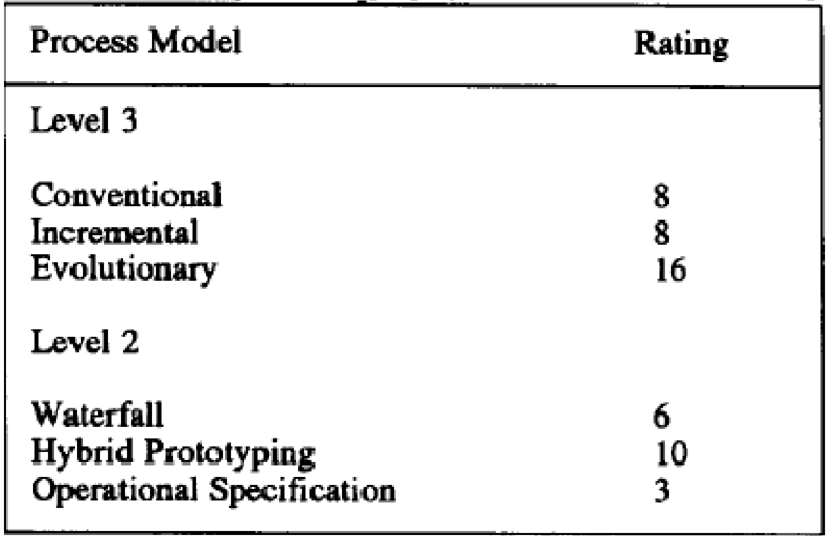
Level 2 process models refine the activities performed in each time segment further and address the different representation of the project (like requirements, design etc.) [Alexander & Davis, 1991] identifies Waterfall, Hybrid prototyping and Operational specification as examples of level 2 models.

At level 1, the process models specify the software methods and tools to use in order to perform the activities described by level 3 and level 2 models. This level of abstraction is discussed further in [Davis, 1990].

As an example, [Alexander & Davis, 1991] examines the selection of an appropriate process model for the project, Prototype Ocean Surveillance Terminal

(POST) using the selection methodology proposed in the paper. The project is characterized using the criteria and then rated against each process model. The results of the evaluation are shown as follows:

Table 8: Process Model ratings for Example project [Alexander & Davis, 1991: p528]



Process Model	Rating
Level 3	
Conventional	8
Incremental	8
Evolutionary	16
Level 2	
Waterfall	6
Hybrid Prototyping	10
Operational Specification	3

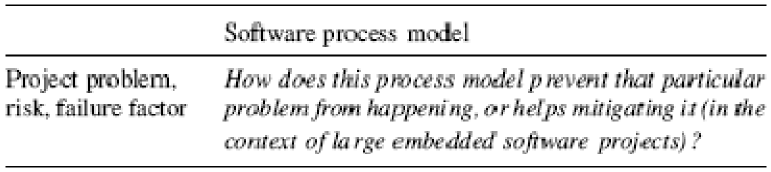
As shown in the table, at the top abstraction level, an evolutionary model is considered the best choice for a process model and at level 2, the hybrid prototyping. The selection criteria described in this paper is certainly not a finite characterization for a project or a process. It does not take into consideration the project technology (web sites, embedded, application) or outside influences like (ISO or FDA). Also, the process is good for selecting the best process for a particular project, but does not provide plans to tailor it for the project. Nonetheless, the selection methodology described in [Alexander & Davis, 1991] is a quantifiable method for process selection that can lead to repeatable and consistent results and is a good basis for this thesis.

3.5 Problem-Based Software Process Selection Model Geared for Embedded Systems

[Kettunen & Laanti, 2004] presents a systematic approach to selecting a software development process model under certain project conditions. Their study is specifically geared towards market driven development of embedded software for telecommunications products. [Kettunen & Laanti, 2004] argues that the software models used for embedded systems are variations of "pure" models that focus solely on the software aspects. Embedded systems often come with certain problems not related to pure software project development: hardware dependencies, understanding interdisciplinary product application domain knowledge, knowledge of the different embedded system types etc. The main concerns in embedded systems development lie not in software construction but in the related systems and hardware engineering issues. As such program managers for embedded systems development must address these problems when choosing an appropriate software process model for the project.

[Kettunen & Laanti, 2004] constructs a software process selection matrix based on project problems and proposes it as a practical aid for systematically selecting an appropriate process. The matrix defined by Kettunen et al. provides comparative analysis of different process models with respect to problems and risk factors faced in the development of large embedded systems (large in terms of requiring more than tens of person-years of development effort.) The models analyzed in this study have been selected through a literature survey and include both traditional and modern process models – waterfall, incremental, spiral model, RUP, FDD, ASD, XP and 'hacking'. Problems and risk factors from well known investigations form the rows of the matrix and are grouped with respect to the area of impact or occurrence in the project life cycle (project initiation, execution etc.) The matrix can be used to analyze alternatives by comparing how each model responds or behaves to a predominant problem, or by evaluating how certain problems are handled by different process models. Table 9 shows the arrangement of the selection matrix as proposed by Kettunen et al.

Table 9: Selection matrix structure [Kettunen & Laanti, 2004: p589]



	Software process model
Project problem, risk, failure factor	*How does this process model prevent that particular problem from happening, or helps mitigating it (in the context of large embedded software projects)?*

To exemplify the usability of the selection matrix, [Kettunen & Laanti, 2004] applies the selection matrix to some of Nokia Group’s past projects that have encountered problems due to an inappropriate selection of the development process. The authors introduce the problem with each project and make suggestions by mapping the problems to the matrix. This helps them determine which process would have been better suited to handle the issue. For example, during the development of Project X, certain requirements are known earlier than others. A waterfall method, where all the requirements have to be known before any further development is done, is not appropriate for this project. The authors map the given problem to the known ones in the matrix like “Unclear project objectives”, “Incomplete requirements/specs” and “Poor requirements management” and conclude that incremental development of the project by accommodating partial requirements at later stages in the cycle would prove a better option.

The use of the selection matrix is not limited to determining the optimal process for a project. As highlighted by [Kettunen & Laanti, 2004], the matrix can also be applied to process models currently in use to evaluate weaknesses and determine future mitigation strategies. Furthermore, it can be used as a tool for training personnel and identifying potential problematic areas in projects.

The authors acknowledge that no one process model can handle all problems faced in the project. For most companies a hybrid model addressing different problems and characteristics of a project may often be the best choice. However, [Kettunen & Laanti, 2004] provides a very comprehensive set of heuristics to evaluate different life cycle behaviors with respect to project problems, and characteristics. The advantage of their selection matrix is its systematic composition that focuses on isolated project problems and risk factors. A project's characteristics can be directly applied to the known problems in the matrix and an evaluation of the process models can be made with respect to the problems. Kettunen et al. propose that a quantitative analysis can be performed by rating the problems and the solutions each method would provide and calculating averages or weighted averages for each process model.

3.6 Comparative analysis of the proposed selection models

A common thread among the various selection models proposed over the years is the idea of evaluating processes based on an abstract view of the project from one dimension. For example, the selection process developed by [Kettunen & Laanti, 2004] abstracts common development problems and performs the selection based on how various processes address the problem. [Davis, Bersoff & Comer, 1988]'s methodology hinges on when different processes meet the users' evolving needs. While evaluating the process models from one particular angle and focusing on a specific characteristic addresses issues related to it, the decisions taken may serve to fail in other areas. For example, if a team decides to focus on just meeting user needs quickly and neglect documentation or design, future versions of the software may end up being much harder to manage. Contrary to this idea, Basili's characterization models and Alexander et al.'s selection criteria provide means to assess and match the processes to the project by taking various aspects of the project into consideration. This allows the project manager to be able to address issues in various management domains and not be limited to any one particular viewpoint.

[Basili, 1985]'s idea of developing models of the project to help understand some characteristic of the development provides an abstract visualization of the project with respect to that characteristic. The quantitative evaluation methodology proposed by Basili has its merits in the continual evolution and improvement of the process through data collection and post-mortem analysis. However, the definition of the project goals and evaluation metrics rely solely on management skills. A skilled manager is able to identify what goals of the project need to focus on and what metrics can be used to assess the response to these goals. Unskilled managers may misplace project priorities causing the project to move in a wrong direction. Basili's evaluation methodology does not define concrete means to select a lifecycle and leaves it the user to identify which process best satisfies the goals of the project. [Basili & Rombach, 1987]'s tailoring process augments the selection by identifying areas where the process impacts the development and how they can be improved. The continual improvement procedure developed by Basili and Rombach allows the process to keep pace with the ever growing technology and market needs and is an essential consideration for any selection methodology.

As mentioned earlier, [Alexander & Davis, 1991]'s selection criteria covers a much broader scope of evaluation than other selection methodologies. This type of selection assesses the project and processes from a unified perspective. In particular, Alexander et al.'s selection methodology provides direct mapping between what the process satisfies and what the project needs using the applicability matrices created using the criteria. Although, the mapping identifies what the process satisfies and where it doesn't meet the project goals, it fails to identify areas where application of the process can impact the project negatively. For example, the Waterfall process is applicable for projects with stable requirements. However it increases the cost and seriously impacts the project development time if applied to projects with chaotic or unsteady requirements. In addition to that, the applicability matrices assume that each criterion defined has the same priority from the management point of view. This idea fails to address projects that put more emphasis on delivering partial requirements than having accessibility to the users, for example. Nevertheless, the direct mapping between process characteristics and the project characteristics finds the most suitable process for the project and also identifies areas for improvement where the process fails to satisfy the project goals.

[Davis, Bersoff & Comer, 1988]'s alternate life cycle comparison introduces some of the primary management concerns like cost increase, late deliveries, failure to address user needs in time, inadaptability of the process and the general inappropriateness of the process selected. It graphically represents a process' response to the user's evolving needs. Although, the methodology fails to address other management aspects like available resources, the graphical representation provides visual insight into how the process can be improved to satisfy project goals.

[Kettunen & Laanti, 2004]'s methodology identifies general problematic areas encountered during project development. Kettunen et al. introduce a selective application principle where a set of problems that best relate to the project needs is selected and used for evaluating various processes. However, the proposed methodology fails to address how the problems can be resolved in the current process.

There are advantages and disadvantages to using any selection model. Some provide concrete means to match process to project but fail to consider management priorities while others identify improvement areas but are not systematic enough to determine which process best suits the project needs. This thesis will build on this work by attempting to incorporate the advantages provided by these various methodologies and to address the weaknesses related to each.

CHAPTER 4
Approach

In order to understand the relation between project, process, teams and tools, semi-structured interviews were conducted with representatives from a range of real project teams spanning a number of application domains. This section describes the interview process, summarizes each interview and describes common themes identified in different projects.

4.1 Method

A qualitative approach was taken to understand the relation between teams, tools, projects and processes. Nine participants from different teams were selected to be interviewed based on their expertise in various application domains (e.g. desktop, systems, embedded, and tools). The interviews were conducted with a manager from each team. When a manager was not available, a member of the team who is familiar with the development process was requested to participate. Each interview was 45-60 minutes in length.

The interview format was a composition of semi-structured questions focused on learning about the project characteristics, the team structure and roles, the tools used for development, the management of the project life cycle, where the requirements came from and the development process employed. The questions were geared towards understanding how the project characteristics impacted the software process adopted and what tools were used to support it. Participants were encouraged to discuss their team structure and the process workflow from inception (requirements elicitation and analysis) to release and maintenance. Enquiries were made to understand the level of involvement with the customers and how the team responds to delays in project schedule, discovery of defects in testing and operation, introduction of new requirements, and need for quick releases. The interview protocol applied for this study is detailed in Appendix B.

4.2 Participants

Table 10 presents an overview of the teams of participants interviewed for this thesis. The teams differ in the application domain, the team size, the process employed, and the length of project development.

Table 10: Teams of interview participants

Team	Date	Time	Application Domain	Team Size	Development Process	Project length / Frequency of external releases
A	12/12/07	3:03 p.m. – 3:54 p.m.	Multi-architectural client-server application	4 developers, 2 QA, 1 test automation, 1 program manager	Spiral Model with Waterfall approach taken within each spiral	1 year – 18 months
B	12/13/07	9:37 a.m. – 10:25 a.m.	Medical Embedded Systems	8 developers, 12 testers	Spiral Model	5 years with 9 month iterations
C	12/17/07	4:00 p.m. – 4:50 p.m.	Internal Tools development	7 members, 1 program manager	Ad-hoc development regulated by request for changes	1 week to 1 month to implement a change
D	12/18/07	3:00 p.m. – 3:48 p.m.	Project Managemen t Software	5-7 people per team	Agile	7-8 weeks external iterations (timeboxes), 2 weeks internal iteration
E	12/20/07	4:02 p.m.	3D Desktop Application	20 developers and 7 testers	Waterfall with short	Generally 1 year

		– 4:50 p.m.			internal iterations	
F	1/11/08	2:05 p.m. – 3:07 p.m.	File Systems	15 developers and 6 testers. The testing team is situated in China and Ireland	Timebox model	2 week iterations
G	2/4/08	12:01 p.m. – 12:43 p.m.	Middleware solutions	30 developers, 5 testers and 1 program manager	Waterfall with very short external iterations	Depends on the defects, but anywhere between 1 week to 1 month
H	2/8/08	10:12 a.m. – 10:55 a.m.	Embedded Systems	3 people in the immediate team, 20 on site and 500 for the whole platform	Iterative development	6 months to 1 year
I	2/8/08	3:03 p.m. – 3:47 p.m.	Document management software	3-5 people	Formal requirements analysis, informal development	3-6 months for some projects, 6-12 months for others.

4.3 Interview Summaries

The following sections summarize the interviews conducted and attempt to answer the following questions for each team:

- *What type of software is developed by the team?*
- *How do the requirements come in?*
- *What development life cycle is used for project development?*
- *How did the project characteristics influence the process?*
- *How often is the project released?*

- *How often do the requirements change?*
- *What tools are used in the development of the software?*
- *What is the level of customer involvement in the project?*

4.3.1 Interview A

Team A is a software applications group in a medical device company. Their product is a multi-architecture application (client–server, web services, web applications, and service oriented architecture) that interfaces with their “flagship” device by automating user input. A few versions of the product have been deployed to a number of customers for a few years. The requirements for the product come in three ways: changes to the device, large scale changes in the market and defects and issues in the current versions. However, the requirements are reasonably stable throughout the development period.

> *Participant A:* “The market can’t accept [new versions] unless we can reduce the impact to our customer when we have a new feature.”

The organization is heavily regulated by the FDA and as such the organizational standard requires much documentation to be produced in order to demonstrate software quality. The project uses Spiral modeling where each spiral ends with a concrete product release and lasts about 1 year to 18 months.

> *Participant A:* “[Each spiral signifies that] we’re not developing a single product and putting it into maintenance mode; we’re developing the product, and our work continues on with the next spiral, which is our next better version [reflecting] what we learned from the previous spiral.”

Within each spiral, a traditional Waterfall development process is applied, consisting of analysis, planning, requirements, design, unit test, integration, verification and validation. Unlike the traditional waterfall methodology though, there are additional "tollgates" added to the process in terms of risk analysis. Depending on the risks and complexity identified during the analysis phase, proof of concept prototypes or "spikes" are created to clarify the requirements. Due to the heavy emphasis on the regulatory and safety aspects in this market space, a process called FMECA (Failure Mode Effects Cause Analysis) is used to perform a full risk analysis of the project. The project is fairly complex being a compilation of a multitude of different technologies and architecture. Both C++ and C# are used for project development with PL/SQL and Oracle backend. A number of third party tools are used during project development from managing the requirements documents (RequistePro) to source control (Visual SourceSafe) to automating the build process (Visual Build Pro) and running automated functional tests (Rational Robot). There are some custom tools built to "tie the whole process together". Bugs are tracked in a defect database called the Issue Tracker system. The team is made up of four developers, two QA testers, one person for test automation and the project manager.

4.3.2 Interview B

Team B develops embedded systems for a medical device in the therapeutics industry. They work with three blood protocols at the same time (therapeutic plasma exchange, red blood cell exchange, and white cell collection) with a team of 8 developers and 12 testers working separately from each other. The project requires five person years of development effort and heavily focuses on code reuse in order to eliminate duplication. Unfortunately, reuse of common code among protocols has sped up the development process while the testing team does not have the same sort of mechanism. This causes the project to be delayed in the testing phase more often than not. Due to the length of the project being so long, the development is broken down to 9 month release cycles. The protocols allow the project to make a natural separation. For every protocol, the product goes through three phases of release. The project is initially released for clinical trials to gather user feedback and perform validation on the project. This release is the first human use of the project. An updated version of the project goes into more routine use with less assistance from the project owners followed by a production quality release. The development life cycle builds heavily from the Spiral development model starting with requirements elicitation from the project stakeholders including clinical and marketing personnel. The result of this step is a requirements specifications document. During the requirements analysis, the requirements are split between the testers and the developers. The developers manage the software control requirements and the testers maintain the safety requirements. From there, screen mockups are developed and tasks are assigned to developers for implementation. The Issue Tracker system is used for describing and tracking the requirements. It is also used to track defects in the system and the progress of the project. Initially as the software is being developed, it is not available for QA testing. Before the software reaches the “feature complete” stage, the project is released regularly to the testing team. The QA team produces a requirements traceability matrix that tracks the requirements and matches them to the tests. The FMECA protocol described in the previous section is applied for Team B’s project as well where all the failure modes are examined and tested by QA personnel. The lack of communication between the developers and the testers has, in the past, resulted in several “non-issues” (issues that turned out to be irrelevant) to be written against the project. Once the feature complete stage is reached the team moves on to develop the next protocol while handling defects found by the QA team in the current build. After the system has been released, a post mortem analysis is performed to investigate areas of improvements in the system and process. The system is object oriented and the UML is used heavily to show interactions between various high-level modules. The sequence diagrams, for example, show the timing of the tasks being executed. Individual development tools are not standardized and left to the developers' preference. The developers are free to use any type of editor or debugger but they have to adhere to a common build technique. The overall build system for the project is standardized to control the release of the product. The developers built a logging and test harness into the device in order to increase the efficiency of testing it. The project is heavily regulated by the FDA and ISO. These bodies make sure the requirements are traceable to the design and the quality of the system is maintained. The FDA also looks at how field issues are being handled by the project.

4.3.3 Interview C

Team C is comprised of 7 people, including the manager, who are responsible for the development and maintenance of tools used by Company C's embedded systems development. Company C is a producer of telephony applications with a product of nearly six billion LOC and 150 developers working in the code base. Because of the scope of the project, a robust set of centrally managed tools is required.

> *Participant C:* "We own all the tools the developer use to set up and deliver their code changes. We own the processes that put the software under the media so that you can install it on the switch out in the field."

The tools are legacy internal applications of the company and are currently mostly in maintenance. Development of new features is guided by a Modification Request system (MR system). The MR system is a change management system Company C uses to document changes in their software as well as the tools. A review board (MRRB) prioritizes the MRs currently assigned to the project. The software process itself is more ad-hoc because of the age of the tools. With 20 years of legacy code and developer's familiarity with the code, requirements traceability is not as rigorous as the actual software of the company. The notion of the life-cycle begins with a modification request which is analyzed and documented before implementation. A mix of technologies is used for development of these tools, and the team is currently looking into investing in transitioning to more sophisticated IDEs (Eclipse). The tools form a development environment for assisting product development through four different phases: Setup (the files that are to be edited are copied to the developer's workspace), Inspect (once the code has been changed, compiled and tested a number of inspectors look through the code to make sure it is correct), Move Workspace (where the code is approved by a number of inspectors) and Report Base (delivers the code changes to the delivery area). Developers use a number of underlying tools during each phase to assist the development (e.g., tools to help merge changes in the code). Another set of tools exist for integrating the code and building the code daily and bi-weekly. Every two weeks, the last stable daily build is run through sanity tests and then handed off for functional testing. The tools are minimally documented in man pages and websites which explain the purpose and use of the tool. The production tools are centrally managed in a bin where changes to the tool are delivered and automatically picked up. The entire development process is periodically audited by the ISO and the MR system is used during these audits to explain changes. These tools are used on a daily basis by 100-150 developers company wide. Testing is mostly automated and run on a daily basis, while sanity tests are run bi-weekly. Customizing third party tools is costly and hence Company C depends on these home grown tools for supporting their product development.

4.3.4 Interview D

Team D is a producer of Agile Project Management software. Their product allows software development teams to track requirements, tasks, test cases and defects. The company is small with a total of 20-25 people including testers, documenters, developers and managers. The team itself is composed of 5-7 people co-located to facilitate team communication. However, they have around 100-300 customers along with nearly 10,000 independent subscribers. Being the proponents of Agile principles, the team practices Agile development themselves. The workflow of a typical development cycle starts with the product manager creating a backlog of stories and prioritizing them. Product managers are in constant communication with the customers to gather requirements and best practices for the project. Acceptance criteria are then defined for the stories using test cases. Tasks are created for the stories and assigned to an iteration. Each story is then implemented by writing unit test cases and completing associated tasks. A timebox model is used for development.

> *Participant D:* "It's important to get something out in each time box. […] The timebox is more important than the functionality or the team resources, so it always ensures that the iteration is short."

There are two timeboxes that Team D adheres to, an internal timebox of 2 weeks and an external release timebox of 7-8 weeks. Both code and tests are written in parallel where the testers write the test infrastructure and the developers write the code. Within 2 weeks, the code starts getting integrated and regularly tested. Team D uses a continuous integration system that builds and tests the software after every check-in. Nightly builds compile and test the software using a wide variety of tests. Performance metrics like code-coverage, code-consistency and other tools are also applied during these builds to measure and report the software's performance. Both automated and manual testing is performed on the project. Mostly automated tests are preferred but manual testing allows Team D to weed out usability defects. Defects from development and maintenance are managed throughout the life cycle. If a defect is identified internally during an iteration, it is fixed immediately and a test case is written against it for regression testing. Defects found in the field are relayed through a CRM (Customer Relation Management) system and are fixed and deployed by the weekend. Automatic notifications are sent to the customers regarding the fix through the CRM system. The project management software developed by the company is used throughout the project lifecycle along with a number of third party tools for development and maintenance like JUnit, Subversion, Clover, Eclipse, IntelliJ.

4.3.5 Interview E

Team E is a 3D modeling and virtualization team in a medium sized company. The team is responsible for development and testing of a 3D modeling desktop application that is released to the general public. The product manager is in constant communication with the customers eliciting requirements and gathering feedback for the current versions. The project is developed using the traditional Waterfall approach that has been “broken down” towards more Agile techniques by implementing smaller iterations. Each iteration of the project produces testable and operational code that is released internally for quality, usability and internal beta testing. The development begins as the product manager gathers requirements from the customers. The project technical lead takes the ideas and characterizes them into development items. These items are evaluated by the development team and assigned as features to be implemented or modified. The project manager of the team then develops a timeline for the features and estimates the overall cost, effort and status of the project for the whole development period. Design documents are written once the estimates are drawn. The overall effort is broken down into pseudo iterations and at the end of each iteration a release is delivered to QA for testing. Progress is tracked using weekly status meetings. The QA team assists the development effort by building automated tools and creating a test harness for the project. The team utilizes a number of custom tools built around their source control that assists in the development of the project. When a check in is made, a notification is sent to another developer in the team who goes through the code and verifies its correctness. There are defect tracking databases that are used by the QA on a daily basis. Builds are run with every check in and minimal testing is performed with each build. The full automated test framework is run once the build is stable. There are a number of automated testing tools that have been tailored to focus on testing each component of the project independently – the scripting engine, the UI, etc. Throughout the project development the requirements aren’t very rigorously traced. There are times when the design and development moves away from the original requirements. The requirements themselves aren’t very easily accessible for the members of the team. The nature of the project being 3D graphics software doesn’t allow for complete automated testing, so some level of manual testing is also involved. The team also makes use of a number of off-the-shelf tools like a source control system, a defect tracker, and UI testing systems and develops custom tools to tailor them to the process.

4.3.6 Interview F

Team F is a file systems development team in a large world wide company with team members in other states and countries. The team is composed of 15 members, development and management, and 6 testers. The roles on the team are decided based on their expertise in certain areas. There are members who specialize in certain areas of the code, while others are general programmers. Larger projects have geographically separated teams who are responsible for testing the projects. The ratio between developers and testers is 3:1. Team F’s testing team is situated in Ireland and China. The communications between the various team members is facilitated through phone conversations at least once a week and an annual face to face meeting. As the file system is part of a bigger project, the company’s operating system, it is delivered as a part of the OS and not independently. The file system is implemented as part of the OS’s development process, which is structured like a train moving through different project builds. Every two weeks the project is built and integrated. Each release acts as part of the train and the number of cars represents the total number of builds of the product. Each team in the OS project targets a particular build to check their code in. Within the two week window, the team writes the code, tests it and checks it into the software code base. Requirements come in as feature requests from users. During the development of the file system, each developer takes a child of the central gate (a copy of the main code base) to do the development in and then checks the changes back into the parent gate. The test team periodically takes a snapshot of the central gate and runs a battery of tests on the latest build.

> *Participant F:* "The expectation is that the code that goes into the gate of the main project is always stable. Stable as possible."

The code is tested thoroughly before its put back into the code base. And within the two week windows allotted per build, it is tested in a larger context. The gate is locked down when the developers "put back" or check in their code changes. When large changes go into the gates, the system is locked down for some days by the "gatekeepers" to allow integration of changes into the code base. The gatekeepers are a team of five people who manage the stability of the code as the changes are committed. There are several levels of checks that make sure that the code going into the gate is correct. Within the team itself, once the code is completed and reviewed, permissions have to be requested in order to be able to put code back into the gate. The permissions are managed through a software system called the RTI tool. At a higher level there is a technical lead that reviews the contents of the code coming in and makes sure that nothing inappropriate gets into the base.

> *Participant F:* "We're very concerned in [this team] that the code remains clean and stable. So we spend a lot of time reviewing code changes and trying to get them as good as possible."

Once the changes are put back into the operating system, the project is released to customers immediately. Outside the two week window, development and maintenance is applied to the code as defects are fixed. When defects come in from the field, they are logged in a system called BugTrack and assigned to one of the teams to resolve. The responsible team gives a priority to the defect and proceeds to fix it. B-Trace is another critical tool used for finding out what's happening inside the kernel, and MDB (Module debugger) is used to analyze why a system crashes. The file system design is influenced by the POSIX standards.

4.3.7 Interview G

Team G develops middleware solutions for electronic program guides, i.e. their software allows vendors the capability to implement TV functionality (e.g. selecting channels, setting the recording etc.) and release it to the market more quickly. The software abstracts the lower level functionality from the set-tops and provides vendors the API to implement TV applications. Team G's development process is mainly driven by requirements coming from the customers and their competition. Regular communication with the customer is maintained in order to understand the requirements.

> *Participant G:* "It'll always be nice to talk to the customer and ask him what you really want, 'is this what you really want?' And I think, it's kind of pessimistic thinking, but 9 times out 10 the customer doesn't know what they want. They pay you to figure that out. And 9 times out of 10 when you figure it out for them, it's not what they want anyway."

These communications are facilitated through on site meetings and demos. The team is made up of 30 developers, about 5 testers and 1 program manager. Communications between the team varies depending on the stage the software development is in. At the start of the project, the developers are busy with the requirements while the testers are busy with the test plans. When the process is in a mature stage and tests find more defects, the communication between testers and developers increases. Team G has adopted a traditional waterfall methodology for product development, starting with requirements analysis, documentation, coding, unit testing and integration. The requirements come in from the customers as vague descriptions called the derivative specifications. Using the specs, formal requirements documents are created and iterated between the team and the customer. The program manager assesses the cost, time, effort and resource availability. The release date is determined by the customer needs. The coding begins once these initial estimates are drawn and testing is done on the stable build. The project release dates vary between one week to one month depending on the type of requirements or defects they need to address. There are three types of releases made to the customer: alpha, beta and final releases. The Alpha release is the initial build of the software which contains the basic functionality required with bugs and incomplete functionality. The beta release is a 90% complete product with a slight number of bugs still present. Finally, product quality builds are released in iterations to the customer. Regarding tools, the philosophy followed by the company is to balance tools with time constraints. Priority is given to completing the requirements. Third-party tools like Bugzilla (defect tracking), Smart CVS (source control) and AraxisMerge (for merging changes) are preferred over custom tools. A dedicated release engineer develops and maintains the build process which is run nightly. The project is a relatively low risk project.

> *Participant G:* “The biggest joke we have here is ‘what’s the worst that can happen if we release buggy software?’ The customer won’t be able to see his TV. Big deal! We can do another release tomorrow.”

4.3.8 Interview H

Company H is an embedded systems development company dealing mainly with cell phone software. Team H is responsible for supplying the latest platforms for the developers and the testers to work on. As such, Team H supports all who work on the platform, both internal and external customers of the company. The immediate team is comprised of 3 members. There are 20 people on site in the team while 500 total for the whole platform. Company H is a customer driven company, so the requirements are drawn from what the customers want, what the carriers want, and what the organization wants. Once the requirements are created, the process goes through a strong design phase where use cases are drawn for multiple scenarios and future aspects are accounted for. The requirements are prioritized and committed to a release version. Each iteration lasts anywhere between six months to a year. A project plan is scoped out with estimates of resources, cost, software and hardware needed. The cost of the project is negotiated based on how many features are to be delivered. This project plan is revisited once the project is in development as the dependencies are not always clear in the beginning.

> *Participant H:* “Maybe the hardware is not available and the chip which had to [be used], got delayed, so [we] have to move to a different target.”

Changes caused by these circumstances result in changes to the requirements and design and the analysis cycle is exercised again. Or the customer can change the requirements at any time. The process is iterative in nature and each iteration has three major milestones: feature complete, code complete and test complete. Feature complete is when all the system interfaces are defined. Code complete is when the implementation is done. However, the project is still open for changes and defect resolution. This stage is followed by a code freeze, where the source control is frozen and no changes can be checked in. Test complete is when all the tests are run and there is an external release deliverable available. The project is released to the QA team daily for testing to be performed on it. There are nearly 10,000 automated test cases and 1000 manual test cases run on the project. The code is stabilized at least two to three months ahead of time. After the code freeze happens full manual testing starts. Automated and unit tests are run even before code freeze. Once a stable build is established, it is released to internal customers as an alpha release and finally to external customers. As Team H's primary responsibility is to interact with customers and make sure the platform is ready for development and the testing to be performed on it, tools facilitating various forms of communication are standardized. Email and phone in general are used often along with electronic tools like VNC, Microsoft Office Communicator, Yahoo Messenger, etc. Automated build scripts written in Perl are kicked-off after every check in. It compiles the code, runs the tests, makes sure the software runs, and reports the result. Nightly automations are not run as it cannot be used to pinpoint where the bug happened. Bugs are tracked using two types of mechanisms: CRM database (Change Request Management) and TeamTrack. Due to the nature of the software being developed by the company, it is hard to find third party tools. Hence, home-grown internal tools are utilized for project development.

4.3.9 Interview I

Team I is a consultant team that mainly works on managing the flow of information for client companies. In most companies, the information flow is maintained in the form of documents or as a representation of rows and columns, like in a database. Team I is responsible for taking the more onerous and nebulous processes and structuring and facilitating them electronically through software. The whole project is paid for by the clients. The team size is not fixed and varies from project to project. Generally the team is about 3-5 people for any given project. If the team grows more than that it is segregated into different teams. There is one dedicated tester assigned to each project who does not write code.

Participant I: "It's very difficult to test what you have developed."

There is no concrete development process defined like the Scrum or XP. The life cycle begins with a very formal analysis where a contract is drawn between the client and the manager. The contract formally defines what problem is being addressed and what is promised to the customer. This document is used as input for the acceptance testing performed near the end of the project to verify that the software does what was promised. Requirements are defined by understanding what problems the client currently faces. They are elicited by talking with the project stakeholders. Unlike regular product companies, a new project domain needs to be learned with every project. The actual development following the requirements gathering and analysis is much less formal with the most detail present in defining interface specifications. As the developers are not co-located, communication between them is minimal and therefore strict interface specifications are defined in order to make sure that the components and modules developed and tested independently integrate smoothly. Even if the modules are being developed by the same person, thorough interface specifications are defined. The documents are brief but provide insight into the project design. Regular weekly meetings are held to discuss the status of the project followed by each of the developers writing short paragraphs of what they will be working on. Testing is ongoing throughout the project life cycle. At the end, formal demonstrations are performed by the whole team on the customer site to gain user feedback and close the deal. Any changes in the requirements made during the development period are a result of incomplete analysis.

> *Participant I:* "The customer himself doesn't know his process well enough to make a computer do it."

The developers directly deal with customers to implement the changes. Every project lasts anywhere between 3-6 months and 6-12 months. Continuous analysis is used to manage requirement changes and development costs of implementing the changes. Mostly third party tools are used like Subversion for source control, SharePoint for managing documents and Bugzilla for managing defects that are not relayed to customers. The tools are selected based on their accessibility over the web by geographically separated developers. Two types of testing are performed on the project: unit testing performed throughout the development period and manual testing executed near the end of the project.

4.4 Interview Analysis

The interviews highlighted several project and team characteristics that heavily influence a development process. For example, companies with heavy documentation requirements tend to use more disciplined processes like Waterfall, while companies that have light documentation requirements benefit from more light weight, agile processes that focus on code and understanding user requirements through testing than documentation. The teams of three out of the nine participants interviewed are required to produce formal documentation at various stages in the life cycle. The level of documentation in the other six teams varies from medium to light depending on the organizational standards. One out of the three teams uses documents as contracts between the customer and the team (requirements documents), between members of the team (interface requirements) and management to developers (status reports). The other two use the spiral development model and produce documentation in discrete phases (requirements, design, test plans etc). Each of these documents undergoes a review to validate its contents before moving to the next phase in the system. These documents are used by FDA and ISO auditors to determine traceability between requirements and design, requirements and test, whether the system is doing what it is supposed to and the safety of the device or application. These external regulatory bodies are responsible for the level of documentation in government regulated companies as projects with insufficient documentation may not be allowed to be introduced in the market. Two out of the three teams that produce heavy documentation are regulated by the FDA, and overall four out of the nine teams are regulated by external bodies like the FDA, ISO and POSIX. The ISO and POSIX standards influence the design of the project more than the documentation.

The frequency of requirements changes is another factor observed in the interviews that influences the development process. The projects of four out of the nine participants interviewed have unsteady requirements that change frequently during the development period. Three out of these four projects have adopted shorter release cycles and an iterative development style to accommodate these changes in the project while the last has a relatively shorter life span than the others (3-6 months). One of the three projects uses Agile process while the other two have modified the Waterfall development approach to support short iterations. Each iteration ends in a tested and operational product that is released externally to customers. On average, two weeks is spent by all three teams on each iteration. Four out of four projects with unsteady requirements have constant communication with the stakeholders to make sure the project is going in the correct direction, i.e. meeting all of the user needs. Feedback from the users is used in subsequent iterations to improve the quality and functionality of the project. Shorter iterations allow constant feedback to be retrieved as customers get a new release every couple of weeks.

Team accessibility and customer accessibility have also been shown to influence the development process as well as the selection of tools for the project. Members of two out of the nine teams are geographically separated. One of the teams is logically divided into development and testing teams who have minimal interaction with each other besides reporting and resolving defects. Defect management systems facilitate these communications as well as provide mechanisms to track their progress. The other team has members who work independently from each other at times. Communications between these members is limited but necessary to make sure the code developed by each is able to work together. This team (Team I) has adopted the use of formal interface specifications to make sure modules and components developed independently are able to work together. Communication between users and developers is important to make sure that the project is consistent with the user requirements. Heavy communication between users and developers are facilitated generally through onsite and face-to-face meetings. Four out of the nine projects use demonstrations of release prototypes to gain feedback from the stakeholders. Three out of the nine projects have established change management tools (Modification Request system, Change Request system, and Customer Relations Management system) to convey the changes in requirements and maintain communication between the customer and the developer. One out of these three also makes use of off-the-shelf tools like Yahoo Messenger and Office Communicator to provide internal customers an open channel for communication.

The following table summarizes the characteristics identified during these interviews and their effect on the project development and the process employed.

Table 11: Project characteristics that influence the development process

Characteristic	Effect on the development process
Level of documentation	Heavy - Requires more disciplined process Light or Medium – More emphasis on testing and understanding requirements through prototyping or coding
Influence from external entities	Enforce standards that the organization must adhere to (e.g. heavy documentation, development procedures or design considerations)
Frequency of changes in requirements (Volatility of requirements)	Unsteady requirements – Shorter release cycles accommodate these changes in the project more smoothly. Requires constant communication with the customers.
Accessibility between team members	Geographically separated teams – Use communication tools to facilitate information sharing between members, or adoption of information sharing procedures (e.g. interface specifications) Co-located teams – More face to face communication.
Communication between developers and users	Facilitated through change management tools along with on-site meetings and face to face communication.

Many of these characteristics drive project teams to tailor their process to meet user's and project's needs. A good example is the traditional waterfall approach adopted by Team G to develop middleware solutions for TV application vendors. The television business is very competitive and in order to meet vendor expectations consistently and release software quicker than the competition, it is imperative to have shorter and faster release cycles. The Waterfall approach does not fully support this requirement and hence was modified by Team G to develop the project in short three to four week iterations. The requirements are gathered for each iteration and the Waterfall life cycle is executed during each iteration. At the end of the iteration, the product is released externally to the customers. This is a common theme observed in at least four out of the nine projects. For most projects, the selected development process does not completely satisfy the project's needs. The teams of four out of the nine participants interviewed for this thesis have modified their development process by fusing elements of another life cycle in order to be able to address their project's characteristics. Team G combined the traditional waterfall approach with a more iterative style of development focusing on releasing the software externally at the end of each iteration. Similarly, Team E's Waterfall methodology has been broken down to adopt Agile's shorter release cycles. But unlike Team G, the iterations are released internally. Team A's spiral model allows successive versions of the project to provide better and more functionality than previous versions. It is also ideal for analyzing risks that come with being a medical application. However, as the requirements are mostly stable and the developers are familiar with the system, a more waterfall approach is followed within each spiral with fewer prototypes. Agile development itself is not a rigid sequence of steps but a combination of flexible principles that prefer results over process. Scrum and XP, two Agile development processes, provide different styles of requirements traceability and project implementation. Team D has combined XP's user stories with Scrum's 30 day iteration and daily status meetings to define a unique process for their Agile project management tool. Adjusting a selected software life cycle to the project's needs is typical for several teams. However the strength of this approach lies in its use of tried principles from different development life cycles to address all of a project's needs.

Tools are used at various stages in the project life cycle to increase the efficiency of product development by automating areas like system integration and functional testing which take a large amount of time and man power if not automated. Nine out of the nine participants interviewed use tools somehow in their product development. The tools most commonly mentioned in these interviews are used for defect management and automated system integration. Nine out of the nine participants interviewed use an automated build and integration system, three of which run continuously with every check in of the code, one is executed every 2 weeks to produce a stable release build and five that run on a daily basis, usually scheduled at night. Third-party defect tracking systems like BugZilla and Team Track are used by three out of the nine teams overall while the other six rely on home-grown tools for defect management. Six out of the nine projects use defect tracking mechanisms and similar tools (Customer Relations Mechanism, Modification Request System, Issue Tracker, Change Request etc) for managing change requests and new requirements as well. Code reviews are usually performed on a face to face basis, however some teams are more concerned with what goes into the software than others, and have developed custom tools to facilitate these reviews. Three out of the nine projects use custom tools to automatically notify other developers about code changes being checked into the source control and prevent check in until the code inspectors have reviewed and approved the code. Nine out of the nine teams use a combination of automated and manual testing. Seven out of these nine run the automated tests with the continuous system integration process while the other two use them for unit testing independent components.

4.5 Summary

This chapter summarized the interviews conducted with participants from a range of real project teams and identified several project characteristics that affect the development process. Analysis of the interviews also identified a tailoring strategy applied by some of the teams that combines more than one development process together to create a hybrid life cycle suitable to the project’s needs. In the next chapter we propose a process selection model that makes use of the characteristics identified in this section to determine the most appropriate life cycle for a project and refines the tailoring strategy to suggest improvements for the selected process.

CHAPTER 5
Unified Process Selection Model

The Unified Process Selection Model (UPSM) is a selection methodology that uses project profiles to match and tailor software development life cycles to a given project. The following sections describe the framework and demonstrate its application with a real software project.

5.1 Profiles

When designing graphical user interfaces, it is essential to know what kind of user will be using the application. The user's characteristics, their knowledge of the application domain, expertise with the computer and the tasks he or she will be performing with the application are necessary in designing a "usable" product. It is also important to know what categories of users (e.g. expert, novice etc.) will be using the application in order to generalize the product interface. User profiles provide a way to isolate particular characteristics of the user to focus on for designing, rather than designing an application for everyone that may not be usable by anyone. [Mayhew, 1999] describes User profiles as:

> "Unless User Interface Designers know the specific characteristics of a population of users (e.g. frequency of use, level of typing skill), they cannot make optimal user interface design decisions for them. The purpose of User Profiles is thus to establish the general requirements of a category of users in terms of overall interface style and approach [p35]."

Software development life cycles are developed to address certain issues encountered by past product development efforts. However, as the software industry and technology evolved over the years, customer and project needs evolved as well. Therefore, when selecting or tailoring a software development life cycle it is vital to know what characteristics of the project it will be targeting. Profiling provides the means to focus the selection efforts to specific characteristics of the project that directly affect major management targets like cost and time. Profiling allows managers to illustrate the project with regards to specific management goals to make "optimal" decisions about the software process and tools needed throughout the development period. The UPSM incorporates the idea of project profiles to define a given project and analyze it with respect to management goals.

5.1.1 Advantages of Using Profiles

Project profiles identify the major risk categories of software development like cost, schedule, problem definition etc. and describe the project with respect to these focal points. There are advantages to typifying projects in this manner:

1. *Profiling allows projects to focus on one management target at a time.* The goals and characteristics of any project span a range of areas where the project can fail or succeed. Trying to identify all the goals of the project collectively may lead to vague objectives and omission of important goals. Abstracting management factors into profiles for evaluation allows project managers to focus on the area in question and mitigate these issues when defining project goals.
2. *Projects can be characterized from more than one viewpoint.* A project might experience failures in unrelated areas of the project cycle (e.g. delays in project schedule and lack of conformance to management standards). Evaluating the project using more than one point of view provides project

managers the flexibility to identify areas where the project is most affected and use the profiles to define improvements.

3. *New product development goals can be defined using project profiles.* New product development goals are usually vague as knowledge about the problem is limited. Project profiles provide means of estimating the project with respect to potential risk areas (e.g. cost, time, team etc.) A concrete set of goals and characteristics can be defined by evaluating the project against all profiles.
4. *Profiles can be used to identify weakest areas of mature project development.* Mature applications tend to have established development processes that address the needs of the application very well. Profiles can be used to formally identify weaknesses in the development process by evaluating it against standard risk areas and looking at points where it diverges from the project goals.
5. *Profiles are not orthogonal to each other.* Profiles are related. Improvements in time management and problem understanding can lead to improvements in the cost of the project.
6. *Profiles provide means to identify improvements in the process.* Profiles can be used to identify the weakest areas of the currently used process by isolating areas it diverges from the project goals. Tailoring the project life cycle by integrating principles from other development processes that have a higher rate of success (i.e. matches the project's goals) can effectively mitigate the vulnerable areas of project development.

7. *Profiles can be used to evaluate tools as well as processes.* Profiling identifies potential improvement goals where tools can be inserted to improve the performance of the process. Tools also serve to form a bridge between techniques adopted from various processes to address project needs.

5.1.2 Types of Profiles

The UPSM defines eight different profiles to evaluate a given project: Cost, Time, Team, Development, Problem, Defect, Market/User, and Management. Each profile abstracts project characteristics with respect to major management goals. These goals were identified by analyzing interviews conducted with participants from real project teams (Section 4) and from related literature. The following sections describe each profile in detail and define a set of evaluation criteria for each profile to determine the goals of the project. Certain criteria can be evaluated from more than one point of view and hence are shared between profiles. Each criterion is assigned a set of values that describes the project characteristics. Some of these criteria and their values are based on the characteristics of software life cycles such as frequency of release, level of documentation etc. Additional criteria are derived from related work in this area (as cited below) as well as from the interviews conducted with members from different project teams for this thesis.

5.1.2.1 Cost Profile

Cost overruns are one of the most visible impacts of software development process failure. The abandonment of the HighBAR project was a direct result of increased project costs [Ewusi-Mensah, 2003]. The flawed decisions adopted for the HighBAR project produced faulty estimates of cost and time. The Cost Profile (Table

12) attempts to identify areas of the project that have an impact on the overall project costs.

Table 12: Cost profile

	Criteria				
Cost Profile	1.	Volatility of requirements	Stable	Unsteady	Chaotic
	2.	Developer understands the requirements	Poor	Fair	Excellent
	3.	Early termination of failed project	Optional	Required	
	4.	Tools to be used for the project development	New	Standard	Established
	5.	Estimated man-months to be spent on the full project	< 4-6 months	< 10-12 months	1-3 years
	6.	Staff allocation	High-low	Level	Low-High
	7.	Response to defects found during operation	Stops development until fixed	Development continuous in parallel	Fixed in next release/iteration
	8.	Size of the team	Small (< 10)	Medium (10-50)	Large (> 50)
	9.	Problem complexity	Simple	Difficult	Complex

Volatility of requirements [Kettunen & Laanti, 2004] – The changes seen in web development projects are much more chaotic than changes encountered for a compiler project. Accommodating frequently changing requirements can quickly increase the cost of the project if not managed properly [Kettunen & Laanti, 2004]. This criterion assesses the frequency of changes in the requirements. Acceptable values are V_{11} = {Stable, Unsteady, Chaotic}.

Developer understands the requirements – Poor understanding of the requirements may imply the need to develop a prototype. This increases the cost of the overall development but also assists in the development of a product that is able to satisfy the user needs. However, when the requirements are well understood, developing a prototype will simply increase the costs without providing any visible benefits. This criterion measures the development team's understanding of the problem being solved. Acceptable values are V_{12} = {Poor, Fair, Excellent}.

Early termination of failed project – Management can determine if a project is failing and decide to terminate it early in order to avoid cost overruns. This criterion measures the necessity of early project termination for failed development. Acceptable values are V_{13} = {Optional, Required}.

Tools to be used for the project development – Selection of appropriate tools for the process and familiarity with them can reduce the overall cost of the project. It is not immediately apparent if a selected tool is suitable for the project. However, the later the failure is detected during the development life cycle, the higher the costs incurred [Prather, 1993]. This criterion assesses the level of tools used for the project as V_{14} = {New, Standard, Established}.

Estimated man-months to be spent on the full project – The longer a project remains in development, the higher the costs of the project are. Scrum, FDD and XP all attempt to address this issue by declaring short iterations. This criterion estimates the number of man-months to be spent on a project. Acceptable values derived from the interviews are V_{15} = {< 4-6 months, < 10-12 months, < 1-3 years}.

Staff allocation [Alexander & Davis, 1991] – Alexander et al. proposes three types of staff profile to adopt for a given project – High-Low (more resources are needed towards the start of the project than towards the end), Level (the amount of people required is even throughout the development period) and Low-High (the resources needed are few towards the start of the project and then increase as it goes into code construction) [Alexander & Davis, 1991]. This criterion measures the staff availability over the entire development period.

Response to defects found during operation – The magnitude of damage caused by defects found in critical operations like life support systems is much more severe than defects found in Microsoft Word for example. The development team has the responsibility to resolve the defect as quickly as possible by either stopping current developmental efforts or assigning some developers to work on it parallel. Either way the cost of development increases, however, minor defects can be demoted to the next release or iteration of the software. This criterion gauges the response to defects found during operational phase of the software using values V_{16} = {Stops development until fixed, Development continues in parallel, Fixed in next release/iteration}.

Size of the team [Cockburn, 2000] – [Cockburn, 2000] states that the size of the team is one of the key players in the management of development costs. The number of people required on the team to address the complexity of the project determines the cost of the effort performed on the project. This criterion assesses the size of the team required for the project. Acceptable values are V_{17} = {Small (less than 10 people), Medium (about 10 to 50 people), Large (more than 50 people on the team)}.

Problem Complexity [Alexander & Davis, 1991] – Large real-time embedded systems tend be more complex than desktop applications because of the additional requirements levied by the hardware interaction. The more complex a project is, the higher is the cost to develop it. This criterion measures the complexity of the project as V_{18} = {Simple, Difficult, Complex}.

5.1.2.2 Time Profile

Delays in the project schedule are often symptoms of impractical planning and poor time management. Small slips in project schedule are generally not a cause of panic, but as poor time management continues, these minor delays add up and result in a project that is months or even years later than initially estimated. Agile practices like Scrum and Extreme Programming address these issues by scheduling fixed time boxes per iteration where something is released at the end of each timebox. Efficient time allocation is one of the prime criteria for successful project management. The Time Profile (Table 13) endeavors to identify elements of project development that directly affect the project time management.

Table 13: Time profile

	Criteria				
Time Profile	1.	Developer understands the requirements	Poor	Fair	Excellent
	2.	Tools to be used for the project development	New	Standard	Established
	3.	Partial Requirements	Not desired	Desired	Critical
	4.	Frequency of external releases	Once	Months	Weeks
	5.	Volatility of requirements	Stable	Unsteady	Chaotic
	6.	Estimated man-months to be spent on the project	< 4-6 months	< 10-12 months	1-3 years

	7.	Developers' knowledge of the problem domain	None	Limited	Expert	
	8.	Application Maturity	New	Demo	In Maintenance	
	9.	Result of defect in the system	Loss of Comfort	Loss of Discretionary money	Loss of Essential Money	Loss of Life
	10.	Response to defects found during operation	Stops development until fixed	Development continuous in parallel	Fixed in next release/iteratio n	

Developer understands the requirements – Misunderstanding the requirements could lead to the development of "wrong" project. Prototyping and interaction with the customers can prevent going down the wrong path and also increase the time needed to develop the product. This criterion measures how well the developer understands the requirements. There are several reasons a developer might have poor understanding of the project; he might be new to the project, or the project is new to the market. A developer familiar with the project will have excellent understanding of the requirements and would probably require much less time for implementation. Acceptable values are V_{21} = {Poor, Fair, Excellent}.

Tools to be used for the project development – The use of new tools for development effort includes a learning curve to familiarize the team with the use of the tool. Standard or established tools that have been used for past developmental efforts would not need a learning curve to be involved in the schedule. The additional training required for new tools can delay the project schedule. This criterion measures the level of tools being used for the product. Acceptable values are V_{22} = {New, Standard, Established}.

Partial Requirements [Alexander & Davis, 1991] – The requirement to deliver a product with partial functionality has become more prominent recently. In response to this requirement the schedule needs to support partial releases to the customer. This criterion measures the requirement for partial functionality. Acceptable values are V_{23} = {Not desired, Desired, Critical}.

Frequency of external releases – Depending on the user requirements, a project may have multiple release dates, or it may need to be delivered fully one time. This criterion assesses the frequency of external releases required for the project. Acceptable values are V_{24} = {Once, Months, Weeks}.

Volatility of requirements [Kettunen & Laanti, 2004] – Changes in the requirements during the development effort can cause developers to backtrack and revise the design and the implementation of the project, thereby increasing the time spent on the requirement. Frequent changes can lead to product schedule delays as the changes are accommodated into the product. This criterion measures the frequency of changes in the requirements with respect to time. The acceptable values are V_{25} = {Stable, Unsteady, Chaotic}.

Estimated man-months to be spent on the project – This criterion assesses the amount of effort needed to complete the whole project in man-months. In Agile techniques, the length of an iteration can be as short as 2 weeks, while for the whole development effort it can take up to 6 months to complete all iterations. The acceptable values are V_{26} = {< 4-6 months, < 10-12 months, < 1-3 years}.

Developers' knowledge of the problem domain [Alexander & Davis, 1991] – The developer's familiarity with the problem domain increases the chances for developing a successful product. The development time decreases as the developer is familiar with the weaknesses of development and can avoid causing typical human introduced errors. This criterion weighs the developer's familiarity with the application. Acceptable values are V_{27} = {None, Limited, Expert}.

Application maturity [Alexander & Davis, 1991 & Brown, McCormick & Thomas, 2000] – Mature applications can exercise proven and tested components from previous development efforts to ensure the "correctness" of the current product [Alexander & Davis, 1991]. In contrast, a new development effort without any priors will require more time to be spent in designing. This criterion evaluates the maturity of the product as V_{28} = {New, Demo, Maintenance}.

Result of defect in the system [Cockburn, 2000] – An airplane control system, where defects in the system can potentially result in loss of life, would require more time to be spent on development, improving security and stability, compared to a desktop application to open compressed files which results in loss of comfort. This criterion determines the criticality of the problem being solved. Acceptable values are V_{29} = {Loss of Comfort, Loss of Discretionary Money, Loss of Essential Money, Loss of Life} [Cockburn, 2000].

Response to defects found during operation – Defects found during product operation can increase the time of development as the development team tries to address them. The more severe the problem is, the more it will impact current development. This criterion measures the impact of critical defects found in the field. Acceptable values include V_{210} = {Stops development until fixed, Development continuous in parallel, Fixed in the next release/iteration}.

5.1.2.3 Team Profile

Team structure and communication between the members of the team is essential for Agile programming. However, some Agile practices do not scale very well for large teams (size > 50). In order to select and tailor the software development life cycle for the needs of the team, it is necessary to understand the team. The Team Profile (Table 14) identifies the characteristics and behavior of the team.

Table 14: Team profile

	Criteria				
Team Profile	1.	Size of the team	Small (< 10)	Medium (10-50)	Large (> 50)
	2.	Location of the members	Co-located	Within walking distance	Separated geographically
	3.	Frequency of Team communication	None	Weekly	Daily
	4.	Code ownership	Individual	Class-based	Team-based
	5.	Frequency of communication between QA and Developers	Regular-Seldom	Level	Seldom-Regular

Size of the team [Cockburn, 2000] – Large teams require more formal means of communication and management when compared to small teams. This criterion measures the size of the team in terms of V_{31} = {Small (less than 10 people), Medium (about 10 to 50 people), Large (more than 50 people)}.

Location of the members – Agile practices are a proponent of co-located teams. Members situated together have an advantage of quick communication that is essential to Agile development practices. It is harder for geographically separated teams to follow this principle. This criterion assesses the accessibility of team members. Acceptable values are V_{32} = {Co-located, Within walking distance, Separated geographically}.

Frequency of Team communication – Traditional development processes require less frequent communication between team members than the modern practices. Communication between members of team establishes the status of the project and facilitates knowledge sharing. However, constant meetings are detrimental to development progress. This criterion assesses the level of formal communication like team meetings between team members. Acceptable values are V_{33} = {None, Weekly, Daily}.

Code ownership – Joint ownership of the code as advocated by Extreme Programming has its advantages and disadvantages. Members of the team will not have to depend on others to edit the code as they can make the change themselves. However, joint code ownership can result in no-code ownership or one person code-ownership as described by [Palmer & Felsing, 2002]. A project manager must decide what level of code ownership should be allowed within the project. This criterion measures the level of ownership assigned to the team or individual. Acceptable values are V_{34} = {Individual (one person is responsible for some classes), Class-based (the ownership is decided by the classes in the object model), Team-based (the team has collective ownership)}.

Frequency of communication between QA and Developers – The involvement of the QA team members varies during the development life cycle. With the waterfall approach, the testing starts once a stable code base has been established, and for XP, test cases are implemented before the code is even written. For the former, the communication between QA members and the developers is more towards the end of the project than the start, while for the latter, the communication is heavier during the start of the project. This criterion gauges the frequency of communication required between the quality assurance and the development. Acceptable values are V_{35} = {Regular-Seldom (the communication is heavier during the start of the project than the end, e.g. automated test engineering), Level (same throughout the development life cycle), Seldom-Regular (the communication is more towards the end of the project)}.

5.1.2.4 Development Profile

Development profile spans the entire development life cycle and focuses on issues directly affecting the development effort for the product (e.g. visibility of progress, developer's knowledge of the problem being solved etc.). Development Profile (Table 15) defines the criteria specific to the project development.

Table 15: Development profile

	Criteria				
Development Profile	1.	Developers' knowledge of the problem domain	None	Limited	Expert
	2.	Developer's S/W Experience	Novice	Experienced	Expert
	3.	Developer understands the requirements	Poor	Fair	Excellent
	4.	Communication between developer and user	Seldom	Regular	Continuous
	5.	Tools to be used for the project development	New	Standard	Established
	6.	Tracking Progress	Through milestones	Feature completion	Iterative releases
	7.	Frequency of	Seldom	Regular	Continuous

		Inspections			
	8.	Response to defects found during operation	Stops development until fixed	Development continuous in parallel	Fixed in next release/iteration

Developers' knowledge of the problem domain [Alexander & Davis, 1991] – Familiarity with the application lends itself to quick development. Less time is spent on understanding requirements and common mistakes are avoided. The developer's knowledge of the problem domain can stem from being a user of the application or being a developer of similar problems [Alexander & Davis, 1991]. This criterion measures the level of knowledge the developer has about the problem domain. Acceptable values are V_{41} = {None, Limited, Expert}.

Developer's S/W Experience [Alexander & Davis, 1991] – This criterion assesses the developer's general knowledge of the programming environment, language and tools to be used for development. Experienced developer is able to adapt to the environment and perform well during the development effort. Novices would need time to adjust. Acceptable values of this criterion are V_{42} = {Novice, Experienced, Expert}.

Developer understands the requirements – Understanding requirements is essential to developing the right product. In new product development, the developer might not be too familiar with the requirements, while mature project's requirements are well understood. This criterion evaluates the developer's understanding of the requirements of the project. Acceptable values are V_{43} = {Poor, Fair, Excellent}.

Communication between developer and user – Constant communication between the developer and user facilitates problem understanding and provides regular progress reports to the users of the application. For mature applications, however, constant communication may not be as desired as the developers are already familiar with the application and it will be cost and time consuming. This criterion measures the level of communication between the developer and user. Acceptable values are V_{44} = {Seldom, Regular, Continuous}.

Tools to be used for the project development – Evolving technology provides means to automate several manual and repetitive tasks to increase productivity and allow development time to be managed well. Established or standard tools are familiar and require training, however, legacy tools are harder to support. This criterion measures the level of tools used for the development effort. Acceptable values are V_{45} = {New, Standard, Established}.

Tracking Progress – Progress of project development can be monitored through deliverables and milestones. Milestones can be set in the form of documentation or status reports. For some projects, feature completion demonstrates the progress of the project while iterative releases themselves display visible progress. This criterion determines how progress is tracked for the project. Acceptable values are V_{46} = {Through milestones, Feature completion, Iterative releases}.

Frequency of Inspections – Inspections of the design and code early on in the development can prevent observable errors to be entered into the code. Continuous inspections, however, can take up significant time from the development effort. This criterion measures the level of inspection performed for the project. Acceptable values are V_{47} = {Seldom, Regular, Continuous}.

Response to defects found during operation – Defects found in the field during operation can be handled by either stopping current development (increases cost and time) or by allocating a separate team to handle the defect (requires additional resources). This criterion assesses the response to defects found during product operation, V_{48} = {Stops development until fixed, Development continues in parallel, Fixed in next release/iteration}.

5.1.2.5 Problem Profile

[Cockburn, 2000] identifies four levels of loss that categorize the types of damage defects in a system can cause – loss of comfort, loss of discretionary money, loss of essential money and loss of life. Loss of essential money or loss of life caused by defects in applications is much more severe than loss of comfort. In order to effectively solve the problem provided by the customers, it is important to understand the nature of the problem. The Problem Profile (Table 16) defines the problem being addressed.

Table 16: Problem profile

	Criteria					
Problem Profile	1.	Problem Complexity	Simple	Difficult	Complex	
	2.	Application Maturity	New	Demo	In Maintenance	
	3.	Result of defect in the system	Loss of comfort	Loss of discretionary money	Loss of essential Money	Loss of life
	4.	Embedded System	No	Somewhat	Completely	
	5.	Interface with hardware	None	Light	Heavy	
	6.	Interface with network components	None	Light	Heavy	
	7.	Interface with humans	None	Light	Heavy	
	8.	Volatility of requirements	Stable	Unsteady	Chaotic	
	9.	Magnitude of changes	Minor	Moderate	Extreme	

	10.	Project Risks	Low	Medium	High

Problem Complexity [Alexander & Davis, 1991] – The complexity of a problem can range from designing a simple calculation algorithm to designing a real-time embedded software control system. This criterion measures the complexity of the problem being solved as V_{51} = {Simple, Difficult, Complex}.

Application maturity [Alexander & Davis, 1991] – This criterion evaluates the maturity of the application and the problem being solved. For mature systems, the domain knowledge from past systems can prove beneficial to current application development. For new development efforts there is insufficient knowledge and the application would require a more evolutionary approach in development [Alexander & Davis, 1991]. Applicable values are V_{52} = {New, Demo, In Maintenance} [Brown et al., 1994].

Result of defect in the system [Cockburn, 2000] – [Cockburn, 2000] lists application criticality as one of the prime principles of process selection. The more critical a system is, the higher the damage undetected defects will produce. For example, a control defect found in the TV channel selection will result in a slight loss of comfort for the viewers; however, if a defect is found in a life support embedded system, it could potentially result in somebody's death. For critical systems a higher visibility of correctness is required. This criteria measures how critical a system is by evaluating the damage caused by defects in the system, V_{53} = {Loss of Comfort, Loss of Discretionary money, Loss of Essential Money, Loss of Life}.

Embedded systems [Kettunen & Laanti, 2004] – Embedded systems require knowledge at the hardware level as well as the software level. Due to many dependencies it is harder to manage embedded systems than it is manage desktop standalone applications. This criterion determines the project's definition as an embedded system using values V_{54} = {No, Somewhat, Completely}.

Interface with hardware – Real time embedded systems require heavy interfacing between the hardware components and the software section, while using software to operate a machine remotely requires relatively light interface. This criterion measures the problem's hardware interface requirements as V_{55} = {None, Light, Heavy}.

Interface with network components – Applications like email clients require interaction with web components like web services. This criterion measures the level of web-interfacing required for the problem. Applicable values are V_{56} = {None, Light, Heavy}.

Interface with humans – Web applications like email systems interface heavily with all types of users while a ground support system being developed by Lockheed Martin might be used by only a select number of trained individuals. This criterion measures the level of interface between the product and humans. Applicable values are V_{57} = {None, Light, Heavy}.

Volatility of requirements [Kettunen & Laanti, 2004] – [Davis, Bersoff & Comer, 1988] explain the user's needs as ever evolving. This criterion measures the frequency of changes in the problem requirements. Acceptable values are V_{58} = {Stable, Unsteady, Chaotic}.

Magnitude of changes [Alexander & Davis, 1991] – Changes in the architecture of the application are much more extreme than changes in the name of the application. This criterion measures the relative size of the change being implemented [Alexander & Davis, 1991]. Application values are V_{59} = {Minor, Moderate, Extreme}.

Project Risks – High risk factors, if not mitigated properly, can increase the complexity of the project. This criterion measures the level of risks present in the project. Acceptable values are V_{510} = {Low, Medium, High}.

5.1.2.6 Defect Profile

Defects are a common affair in the development of any project, regardless of the size of the project. As observed in the interviews with the representatives from real projects, establishing a good defect management system helps mitigate the cost of maintenance and amplifies the quality of the final product delivered. Defects can be observed and addressed at various stages of the development process. The Defect Profile (Table 17) identifies the defect management techniques that would be ideal for the project.

Table 17: Defect profile

	Criteria				
Defect Profile	1.	Defect management	New	Standard	Established
	2.	Magnitude of changes	Minor	Moderate	Extreme
	3.	Frequency of Inspections	Seldom	Regular	Continuous
	4.	Frequency of system integration	Occasionally	Regular	Continuous
	5.	Frequency of test builds	Occasionally	Regular	Infrequent-Frequent

Defect management – Interviews with participants from real project teams highlighted the need for efficiently managing and tracking defects in the project. This criterion measures the state of defect management established for the project. Acceptable values are V_{61} = {New, Standard, Established}.

Magnitude of changes [Alexander & Davis, 1991] – This criterion measures the affect of expected changes to the system. Adding a new sensor to a ground control system is classified as an extreme change while changing the UI screens for a little more flexibility is a more minor change. Acceptable values are V_{62} = {Minor, Moderate, Extreme}.

Frequency of inspections - Regular inspections of the code, design and requirements spot defects earlier in the life cycle; however they also require considerable time to be conducted regularly. This criterion evaluates how often inspection take place within the work environment. Acceptable values are V_{63} = {Seldom, Regular, Continuous}.

Frequency of system integration – Data collected from interviews indicate the necessity of having continuous integration systems that run regularly (e.g. Nightly), continuously (e.g. with every check-in), or occasionally at the developer's discretion. This criterion evaluates the level of system integration performed for the project as V_{64} = {Occasionally, Regular, Continuous}.

Frequency of test builds – Companies with parallel execution of development and test release builds to QA on nearly a daily basis while companies that are working on complex problems might require one build to be tested completely before moving to another. This criterion ascertains the frequency of test builds to QA over the development period. Applicable values are V_{65} = {Occasionally, Regular, Infrequent-Frequent}.

5.1.2.7 Market/User Profile

Similar to User Profiles described by [Mayhew, 1999], Market/User Profile attempts at clarifying the user's role in the project development. The ultimate tester for any project is the user of the product and if the application does not meet the user requirements or the developers produce entirely different software then the project is not usable from the user's point of view. The Market/User Profile identifies who the users are and what level of involvement from the user is required to develop the product (Table 18).

Table 18: Market/User profile

	Criteria				
Market / User Profile	1.	Volatility of requirements	Stable	Unsteady	Chaotic
	2.	User's knowledge of the problem domain	None	Limited	Expert
	3.	User's communication of the needs	None	Limited	Expressive
	4.	Market requirement for the release of fully functional system	Not desired	Desired	Critical
	5.	User understands the requirements	Poor	Fair	Excellent
	6.	Target customer size	Small	Medium	Large
	7.	Target customer business	Specialized	General	Government
	8.	User Feedback	Not desired	Desired	Critical
	9.	User's acceptance of functionality de-scoping	Favorable	Negotiable	Not desired
	10.	Frequency of external releases	Once	Every couple of months	Every couple of weeks

11.	Frequency of access to users	Once a month	On a weekly basis	Continuous

Volatility of requirements [Kettunen & Laanti, 2004] – [Davis, Bersoff & Comer, 1988] explains the importance of tracking and meeting users' evolving needs. This criterion measures the frequency of changes in the requirements as V_{71} = {Stable, Unsteady, Chaotic}.

User's knowledge of the problem domain [Alexander & Davis, 1991] – Novice users are more acceptable to suggestions, but are able to provide less input because of their unfamiliarity with the problem domain. Experienced users are very familiar with the domain and know exactly what they want to address, however, they might be more resistant to changes [Alexander & Davis, 1991]. This criterion measures the user's knowledge of the problem domain. Applicable values are V_{72} = {None, Limited, Expert}.

User's communication of the needs [Alexander & Davis, 1991] – Even if the user is very familiar with the domain, if they are not able to express the needs well, it increases the chance of developers misunderstanding the requirement. This criterion measures how well the user is able to communicate their requirements. Applicable values are V_{73} = {None, Limited, Expressive}.

Market requirement of the release of fully functional system [GSAM, 2003] – A control system designed to pilot planes automatically requires a fully-functioning system to be deployed while web applications may benefit from incremental addition of functionality. This criterion measures the market need to have a fully functional system. Applicable values are V_{74} = {Not desired, Desired, Critical}.

User understand the requirements – Users understanding of what is needed for their problem assists in defining concrete set of features and requirements for the software. This criterion measures how well the customer understands their needs. Applicable values are V_{75} = {Poor, Fair, Excellent}.

Target customer size – A small customer size indicates the project is targeting specific requirements and needs special user involvement. Addressing a large number of companies usually requires a general application that is applicable to more than one environment. This criterion evaluates the size of the target customers as V_{76} = {Small, Medium, Large}.

Target customer business – Customers with private business firms require specific software tailored for their needs, while software addressing the general public or a number of companies in general requires a more generic product. Government projects are usually strictly regulated for security and quality purposes. This criterion determines the target customer business requirements, V_{77} = {Specialized, General, Government}.

User feedback – During the development of new products, or when requirements are not as clearly understood, having constant user feedback becomes a necessity. This criteria measures the level of feedback required from the user as V_{78} = {Not desired, Desired, Critical}.

User's acceptance of functionality de-scoping – For established milestones to be fulfilled on time, sometimes de-scoping of the project functionality or requirement is necessary when the project schedule starts slipping. This criterion gauges the customer's response to such de-scoping to determine whether the system is to be schedule-oriented or functionality-oriented. Accepted values are V_{79} = {Favorable, Negotiable, Not desired}.

Frequency of external releases – Target release dates estimates for a project are dependent on customer release requirements. This criterion measures the frequency of releases required as a result of the user needs. Applicable values are V_{710} = {Once, Every couple of months, Every couple of weeks}.

Frequency of access to users [Alexander & Davis, 1991] – Agile practices may be taxing for companies that have very limited access to customers, as its principles are based on heavy customer involvement. This criterion measures the frequency of access to the customers. Available values are V_{711} = {Once a month, On a weekly basis, Continuous}.

5.1.2.8 Management Profile

Management is an essential player in the planning and organization of the project. Established company cultures have hierarchical organization structures that are optimized for the company needs. In order to support a process that is ideal for such company's projects, it is important to take into consideration the company standards and policies for project development. External entities like ISO, IEEE and FDA provide their own set of standards for commercial products. If the project does not conform to such standards, releasing it on the market is practically impossible. The Management Profile attempts to identify necessary elements of management that are taken into consideration when selecting or tailoring the development process (Table 19).

Table 19: Management profile

	Criteria				
Management Profile	1.	Compatibility with organizational standards	Flexible	Negotiable	Strict
	2.	Influence of external regulatory entities	None	Light	Heavy
	3.	Level of documentation	None	Light	Heavy
	4.	Compatibility with QA Standards	Flexible	Negotiable	Strict
	5.	Compatibility with External standards	Flexible	Negotiable	Strict
	6.	Dependency on other teams	None	Light	Heavy

Compatibility with organizational standards [Alexander & Davis, 1991] – Established organizations require certain quality and management standards to be enforced for all commercial product development. This criterion measures the necessity of conforming to such standards as V_{81} = {Flexible, Negotiable, Strict}.

Influence of external regulatory entities – Strict standards are enforced by regulatory bodies both in U.S. and elsewhere in the world. In order to be a marketable product, certain industries are regulated by these entities to enforce security, correctness and to determine the overall safety of using these applications like medical devices or passenger aircraft. This criterion measures the level of influence these external regulatory entities have on the product development. Applicable values are V_{82} = {None, Light, Heavy}.

Level of documentation – Government regulated industries, for example medical device companies, are required to produce a satisfactory level of documentation demonstrating their conformance to the government standards and the level of security built into the system. Other companies that require conformance to ISO standards or are influenced by external entities are also subject to this requirement. This criterion evaluates the level of documentation that is required for the project as V_{83} = {None, Light, Heavy}.

Compatibility with QA Standards [Alexander & Davis, 1991] – Life support systems require much more rigorous quality assurance execution than a game application. This criterion measures the project's compatibility requirements to the QA Standards as V_{84} = {Flexible, Negotiable, Strict}.

Compatibility with External Standards – ISO and FDA standards are enforced for highly critical embedded device applications that can cause loss of life. Less critical applications do not require such strict enforcement of such standards. This criterion assesses the project's compatibility requirements to standards from external entities as V_{85} = {Flexible, Negotiable, Strict}.

Dependency on other teams – Product development is based on several factors including the technology used for the product, cross-functional teams etc. This criterion measures the project's dependency on other teams like the IT department or marketing as V_{86} = {None, Light, Heavy}.

5.2 Framework

The UPSM framework is founded on [Alexander & Davis, 1991]'s criteria based selection model. [Alexander & Davis, 1991] present a set of twenty criteria to assess a given project and processes. Extending the notion developed by Alexander et al. the UPSM categorizes the evaluation criteria into Profiles. Each profile addresses a management concern and is evaluated through a set of criteria that impacts the profile subject (e.g. cost, time, etc.). Using the profiles and the evaluation criteria, the UPSM produces a list of development processes in the order that best suits the project needs and also identifies areas for improvement in the process. As mentioned in Section 2.4, for the purpose of this study, a subset of the modern software processes is considered: Waterfall, Spiral, Scrum, XP, and FDD. However, the framework is applicable to any number of development life cycles.

Before explaining the framework, it is essential to understand the common notations used by the selection model:

P is the set of profiles selected for evaluating the project. Each profile, p_i where i = 1...8 (e.g. p_1 = Cost, p_2 = Time, p_8 = Management), addresses a particular management concern with respect to the success of the project. The profiles used by UPSM are described in Section 5.1.2;

C_i is the set of criteria associated with a profile, p_i. Each criterion, c_{ij} (e.g. c_{83} = Level of documentation, c_{23} = Partial functionality), assesses the project with respect to the profile subject. The criteria used by UPSM are described in Section 5.1.2;

V_{ij} is the set of values that a criterion, c_{ij}, may take on for the project. Each value, v_{ijk} (e.g. v_{831} = None, v_{832}= Light, v_{833} = Heavy) are mutually exclusive values that are applicable to the criterion, c_{ij}. The values used for UPSM are described in Section 5.1.2;

a_{ijk} represents the applicability of a process to the value, v_{ijk}. The indicators {0, 1, -1} determine how the value, v_{ijk}, applies to the process. 0 indicates that the process is not applicable for the value, 1 signifies that the process does apply to the value, while -1 indicates that the process has a negative impact on project's profile (e.g. a_{111} = 1, a_{112} = -1, a_{113} = -1 for the Waterfall process indicates that the process is applicable for a project where the requirements are stable, but may impact the project's cost negatively, i.e. significantly increases the cost of the project, if the process is applied). The a_{ijk}'s for each process form applicability matrices with respect to the profile and are used to determine the rating of a process model for the project. Appendix A contains the applicability matrices for all the processes considered in this study. Table 20 shows the applicability matrices for the Waterfall process;

Table 20: Applicability matrices for Waterfall process

Cost Profile =

$$\begin{bmatrix} Criteria_1 & v_{1j1} & v_{1j2} & v_{1j3} \\ c_{11} & 1 & -1 & -1 \\ c_{12} & -1 & 1 & 1 \\ c_{13} & 0 & 0 & \\ c_{14} & -1 & 1 & 1 \\ c_{15} & 0 & 1 & 1 \\ c_{16} & -1 & 0 & 1 \\ c_{17} & -1 & 0 & 0 \\ c_{18} & 1 & 1 & 1 \\ c_{19} & 0 & 1 & 1 \end{bmatrix}$$

Time Profile =

$$\begin{bmatrix} Criteria_2 & v_{2j1} & v_{2j2} & v_{2j3} & v_{2j4} \\ c_{21} & -1 & 1 & 1 & \\ c_{22} & -1 & 1 & 1 & \\ c_{23} & 0 & -1 & -1 & \\ c_{24} & 1 & -1 & -1 & \\ c_{25} & 1 & -1 & -1 & \\ c_{26} & 0 & 1 & 1 & \\ c_{27} & -1 & 1 & 1 & \\ c_{28} & 0 & -1 & 1 & \\ c_{29} & 1 & 1 & 1 & 1 \\ c_{210} & -1 & 0 & 0 & \end{bmatrix}$$

Team Profile =

$$\begin{bmatrix} Criteria_3 & v_{3j1} & v_{3j2} & v_{3j3} \\ c_{31} & 1 & 1 & 1 \\ c_{32} & 1 & 1 & 1 \\ c_{33} & -1 & 1 & 0 \\ c_{34} & 0 & 0 & 0 \\ c_{35} & -1 & 1 & 1 \end{bmatrix}$$

Development Profile =

$$\begin{bmatrix} Criteria_4 & v_{4j1} & v_{4j2} & v_{4j3} \\ c_{41} & -1 & 1 & 1 \\ c_{42} & 1 & 1 & 1 \\ c_{43} & -1 & 1 & 1 \\ c_{44} & 1 & 1 & 0 \\ c_{45} & -1 & 1 & 1 \\ c_{46} & 1 & 0 & 0 \\ c_{47} & -1 & 1 & 0 \\ c_{48} & -1 & 0 & 0 \end{bmatrix}$$

Problem Profile =

$$\begin{bmatrix} Criteria_5 & v_{5j1} & v_{5j2} & v_{5j3} & v_{5j4} \\ c_{51} & 0 & 1 & 1 & \\ c_{52} & 0 & -1 & 1 & \\ c_{53} & 0 & 0 & 1 & 1 \\ c_{54} & 1 & 1 & 1 & \\ c_{55} & 1 & 1 & 1 & \\ c_{56} & 1 & 1 & 1 & \\ c_{57} & 1 & 1 & 0 & \\ c_{58} & 1 & -1 & -1 & \\ c_{59} & 0 & 0 & 0 & \\ c_{510} & 1 & 1 & 0 & \end{bmatrix}$$

Defect Profile =

$$\begin{bmatrix} Criteria_6 & v_{6j1} & v_{6j2} & v_{6j3} \\ c_{61} & 0 & 1 & 1 \\ c_{62} & 0 & 0 & 0 \\ c_{63} & -1 & 1 & 0 \\ c_{64} & 1 & 1 & 0 \\ c_{65} & 0 & 0 & 1 \end{bmatrix}$$

Market/User Profile =

$$\begin{bmatrix} Criteria_7 & v_{7j1} & v_{7j2} & v_{7j3} \\ c_{71} & 1 & -1 & -1 \\ c_{72} & -1 & -1 & 1 \\ c_{73} & -1 & -1 & 1 \\ c_{74} & 0 & 1 & 1 \\ c_{75} & -1 & -1 & 1 \\ c_{76} & 1 & 1 & 1 \\ c_{77} & 1 & 1 & 1 \\ c_{78} & 0 & 1 & -1 \\ c_{79} & 0 & 0 & 0 \\ c_{710} & 1 & -1 & -1 \\ c_{711} & 1 & 0 & 0 \end{bmatrix}$$

Management Profile =

$$\begin{bmatrix} Criteria_8 & v_{8j1} & v_{8j2} & v_{8j3} \\ c_{81} & 1 & 1 & 1 \\ c_{82} & 1 & 1 & 1 \\ c_{83} & -1 & -1 & 1 \\ c_{84} & 1 & 1 & 1 \\ c_{85} & 1 & 1 & 1 \\ c_{86} & 1 & 1 & 1 \end{bmatrix}$$

s_{ijk} is a binary indicator of which value, v_{ijk}, applies to this particular project (e.g. s_{111} = 1, s_{112} = 0, s_{113} = 0 indicates that the project has stable requirements, and is not chaotic or unsteady). The indicator 0 implies that the value is not applicable to the project, while 1 suggests applicability. The s_{ijk}'s for the profile form a project characteristics matrix that describes the project from the profile's point of view. The s_{ijk}'s for all the profiles represent the attributes of the project. The s_{ijk}'s vary from project to project and are supplied for every project; and

r_{ij} is the rating assigned to a criterion, c_{ij}, to indicate the importance of the criterion to the project with respect to the profile (e.g. r_{11} = 3 implies that volatility of requirements has a high impact on the project's cost, and hence is very important). The ratings are assigned as 0, the criterion is not applicable to the project, 1, the criterion is of low importance to the project, 2, the criterion is somewhat important to the project, and 3 the criterion highly impacts the profile subject. These ratings are assigned for a particular project and may differ from project to project.

The UPSM rates the various development processes with respect to the project (the highest being the best suited for the project) and determines areas for improvement within the process. The UPSM suggests that improvements can be made by creating a hybrid process by integrating elements from other processes into the selected process, when applicable, by means of tools. The following steps describe the selection model in detail:

1. *Select the set of profiles, P, that best address the goals or concerns of the project* – An advantage of using profiles is to be able to characterize and evaluate projects from different points of view. This step requires the project managers to examine the project's goals and determine what

profiles best relate to the goals or concerns of the project. For example, if one of the goals of the project is to reduce the cost of development, then the Cost and Time profiles should be selected. Ideally, for new project development, it is best to evaluate the project against all the profiles.

2. *For each profile, p_i, determine the r_{ij} by rating each criterion, c_{ij}, with respect to the profile* – [Alexander & Davis, 1991] assumes that each criterion in their criteria-based selection model is of equal importance to the project. However, this is not true for all projects. Some projects might consider the volatility of requirements to be a much higher driver for their costs, than the introduction of tools. This step requires the manager to assess the importance of each criterion, c_{ij}, in the profile, p_i, and rate it on a scale of 0-3, 0 being not applicable, while 3 represents high impact to the profile.
3. *For each value, v_{ijk}, determine the s_{ijk} by indicating whether the value is applicable to this particular project* – This step is similar to Alexander et al.'s process model where each criterion is evaluated against the project needs and assigned a 0 if not applicable, and 1 if it is applicable.
4. *Compute the RATING for each process model using the following formula* $RATING = \sum_i \sum_j \left(r_{ij} \times \sum_k \left(s_{ijk} \times a_{ijk}\right)\right)$; where i = the profile being evaluated, j = the criterion associated with the profile, and k is the value applicable to the criterion. The process model with the highest rating is considered the best choice for the project, followed by the process model with the second best rating and so on.

5. *Analyze* $(s_{ijk} \times a_{ijk})$*'s of each profile,* p_i*, for the selected process model to determine the key areas for improvement* – The $(s_{ijk} \times a_{ijk})$ represents the process' applicability to the project's needs. For example, for a project with stable requirements the $s_{111} \times a_{111}$for the Waterfall process will result in 1, suggesting that the process is applicable for this attribute of the project. Similarly, the areas where the process fails to meet the project's needs are considered the weakest and can be improved.
6. *Suggest improvements by evaluating* $(s_{ijk} \times a_{ijk})$*'s for other processes* – Each process addresses certain projects positively and others negatively. For example, the Waterfall process is good for stable requirements, but does not support frequent changes in the requirements very well, while XP is designed to handle frequent changes, but applying it to stable requirements might increase the cost as a result of providing frequent releases. In this step, the project managers are asked to analyze how other processes handle the weakest areas of a recommended process and assess the cost of integrating mechanisms from other processes into the current process.
7. *Develop metrics to analyze the improvement with respect to the profile* – [Basili, 1985]'s discussions on the goal-question-metric analysis provides the basis for this step. In order to assess the effectiveness of the suggested improvements, it is essential to evaluate its performance for the current project. This step requires decomposing the identified process

improvements into quantifiable metrics that can be used to evaluate future versions of the software.

8. *Analyze and select tools that address the improvement goals* – The $(s_{ijk} \times a_{ijk})$ evaluation provides insight into areas of improvements for the process. One way to facilitate the improvements is through the insertion of tools. As discussed in Section 2.5, it is essential to establish the purpose of the tool, understand its strength and weaknesses, quantify the expected benefits and be able to evaluate the actual benefits against the expected benefits. The tool should be able to implement the process improvements and be able to incorporate data collection for the metrics identified in the previous step.
9. *Apply the process and collect data for the metrics* – In this step, the modified process and tools are applied to the project and data is collected throughout the project life cycle evaluating the performance of the improvements and the effectiveness of the tools. As recommended by Basili, storing this information in a corporate database allows managers from different teams to evaluate their project based on past data [1985].
10. *Analyze the metrics to determine areas for improvement in future versions of the project* – The performance metrics identify areas where the improvements were effective and where corrective measures should be taken. These areas are then analyzed for their impact on the project's success and failure and to determine improvement goals for the next

project. The improvement goals act as input for identifying the set of profiles to be evaluated for the next project.

Software process selection method described by UPSM is a continuously improving phenomenon. No process model meets all of the projects requirements from the onset. The process needs to be monitored over time to determine areas of improvements and changes in relation to the project. As the project evolves, so do the goals of the process. Hence, the application of a continuous improvement strategy provides the means to meet a project's goals.

5.3 Application

As an extension of the interview conducted regarding their process, project and tools, Participant A was asked to estimate the s_{ijk} and r_{ij} values for their project. This section illustrates the application of the UPSM framework for Project A.

1. *Select the set of profiles, P, that best address the goals or concerns of the project* – If one of the goals of the project is to manage development efforts and reduce the cost of the project, then Cost, Time and Development profiles should be selected. Market/User and Problem profile should be selected when the goals of the project deal with addressing user needs and clarifying the problem domain. In order to refine the profile and criteria structures, all profiles where included in the evaluation process for Project A.
2. *For each profile, p_i, determine the r_{ij} by rating each criterion, c_{ij}, with respect to the profile* – Table 21 reflects Participant A's estimation of the project characteristics. The matrices illustrate the rating assigned to each criterion with respect to the profile subject (e.g. cost, time, team, etc.) and the

applicability of the values, v_{ijk}, to the project. Project A is a medical software application in maintenance with a small team and established technology. The current process revolves around managing their resources, addressing the requirements to user satisfaction and facilitating communication among various team members and members from different teams while reducing the cost of the project and developing a product that meets the FDA standards. For this purpose, criteria demonstrating these attributes towards their related profiles are given a higher rating than others. For example, the stability of the requirements and the size of the team are major drivers of the cost and hence are given a high rating for the Cost and Time profile. Also, as the project is in maintenance, the developers are familiar with the problem domain but, in order to meet user goals, the requirements need to be understood fully by the developers before development begins. Therefore, while developer software experience and familiarity with the problem domain is given a medium rating, developer's understanding of the requirements is given a higher rating for the Development profile. Project A is regulated heavily by the FDA and it must produce the required documentation demonstrating the design, quality standards and test results in order to clear FDA audits. Therefore heavy documentation and conformance to organizational standards along with external regulatory standards is required for the project.

3. *For each value, v_{ijk}, determine the s_{ijk} by indicating whether the value is applicable to this particular project* – Project A is a software application that interfaces with embedded software and has been in maintenance for some

years. The environment is established with tools for development, defect management and automated testing. The project team is small and co-located. As the project is in maintenance, the requirements are stable and for the current release of Project A, the changes are moderate. Table 21 illustrates these characteristics for Project A along with others like the critical need for a fully functional product release, user involvement for validation of the project etc. The characteristics are determined by selecting the value, v_{ijk}, of the criterion, c_{ij}, that is most applicable to the project.

Table 21: Project A's characteristics

Cost Profile =

$$\left[\begin{array}{ccccc} C_1 & r_{1j} & v_{1j1} & v_{1j2} & v_{1j3} \\ c_{11} & 3 & 1 & 0 & 0 \\ c_{12} & 3 & 0 & 1 & 0 \\ c_{13} & 0 & 0 & 0 & \\ c_{14} & 2 & 0 & 0 & 1 \\ c_{15} & 2 & 0 & 1 & 0 \\ c_{16} & 3 & 0 & 1 & 0 \\ c_{17} & 3 & 1 & 0 & 0 \\ c_{18} & 3 & 1 & 0 & 0 \\ c_{19} & 1 & 0 & 1 & 0 \end{array}\right]$$

Time Profile =

$$\left[\begin{array}{cccccc} C_2 & r_{2j} & v_{2j1} & v_{2j2} & v_{2j3} & v_{2j4} \\ c_{21} & 3 & 0 & 1 & 0 & \\ c_{22} & 2 & 0 & 0 & 1 & \\ c_{23} & 2 & 1 & 0 & 0 & \\ c_{24} & 2 & 1 & 0 & 0 & \\ c_{25} & 3 & 1 & 0 & 0 & \\ c_{26} & 2 & 0 & 1 & 0 & \\ c_{27} & 2 & 0 & 1 & 0 & \\ c_{28} & 0 & 0 & 0 & 1 & \\ c_{29} & 2 & 0 & 0 & 0 & 1 \\ c_{210} & 2 & 0 & 1 & 0 & \end{array}\right]$$

Team Profile =

$$\left[\begin{array}{ccccc} C_3 & r_{3j} & v_{3j1} & v_{3j2} & v_{3j3} \\ c_{31} & 3 & 1 & 0 & 0 \\ c_{32} & 1 & 1 & 0 & 0 \\ c_{33} & 3 & 0 & 0 & 1 \\ c_{34} & 1 & 1 & 0 & 0 \\ c_{35} & 3 & 0 & 0 & 1 \end{array}\right]$$

Development Profile =

$$\left[\begin{array}{ccccc} C_4 & r_{4j} & v_{4j1} & v_{4j2} & v_{4j3} \\ c_{41} & 2 & 0 & 1 & 0 \\ c_{42} & 2 & 0 & 1 & 0 \\ c_{43} & 3 & 0 & 1 & 0 \\ c_{44} & 2 & 1 & 0 & 0 \\ c_{45} & 2 & 0 & 0 & 1 \\ c_{46} & 2 & 1 & 0 & 0 \\ c_{47} & 2 & 0 & 1 & 0 \\ c_{48} & 2 & 0 & 1 & 0 \end{array}\right]$$

Problem Profile =

Defect Profile =

$$\begin{bmatrix} C_5 & r_{5j} & v_{5j1} & v_{5j2} & v_{5j3} & v_{5j4} \\ c_{51} & 1 & 0 & 1 & 0 & \\ c_{52} & 2 & 0 & 0 & 1 & \\ c_{53} & 2 & 0 & 0 & 0 & 1 \\ c_{54} & 0 & 1 & 0 & 0 & \\ c_{55} & 2 & 0 & 1 & 0 & \\ c_{56} & 2 & 0 & 1 & 0 & \\ c_{57} & 3 & 0 & 0 & 1 & \\ c_{58} & 3 & 1 & 0 & 0 & \\ c_{59} & 3 & 0 & 1 & 0 & \\ c_{510} & 3 & 0 & 0 & 1 & \end{bmatrix} \quad \begin{bmatrix} C_6 & r_{6j} & v_{6j1} & v_{6j2} & v_{6j3} \\ c_{61} & 3 & 0 & 0 & 1 \\ c_{62} & 3 & 0 & 1 & 0 \\ c_{63} & 2 & 0 & 1 & 0 \\ c_{64} & 2 & 0 & 1 & 0 \\ c_{65} & 2 & 0 & 0 & 1 \end{bmatrix}$$

Market/User Profile =

$$\begin{bmatrix} C_7 & r_{7j} & v_{7j1} & v_{7j2} & v_{7j3} \\ c_{71} & 3 & 1 & 0 & 0 \\ c_{72} & 3 & 0 & 1 & 0 \\ c_{73} & 2 & 0 & 0 & 1 \\ c_{74} & 3 & 0 & 0 & 1 \\ c_{75} & 3 & 0 & 0 & 1 \\ c_{76} & 1 & 1 & 0 & 0 \\ c_{77} & 3 & 1 & 0 & 0 \\ c_{78} & 3 & 0 & 0 & 1 \\ c_{79} & 1 & 0 & 1 & 0 \\ c_{710} & 3 & 1 & 0 & 0 \\ c_{711} & 3 & 1 & 0 & 0 \end{bmatrix}$$

Management Profile =

$$\begin{bmatrix} C_8 & r_{8j} & v_{8j1} & v_{8j2} & v_{8j3} \\ c_{81} & 3 & 0 & 0 & 1 \\ c_{82} & 3 & 0 & 0 & 1 \\ c_{83} & 3 & 0 & 0 & 1 \\ c_{84} & 2 & 0 & 1 & 0 \\ c_{85} & 3 & 0 & 0 & 1 \\ c_{86} & 2 & 0 & 1 & 0 \end{bmatrix}$$

4. *Compute the RATING for each process model using the following formula:* $RATING = \sum_i \sum_j \left(r_{ij} \times \sum_k \left(s_{ijk} \times a_{ijk}\right)\right)$. Table 22 demonstrates how the rating is computed for the Waterfall process. The RATING for the process model is determined by adding all the t_{ij} values ($t_{ij} = r_{ij} \times \sum_k (s_{ijk} \times a_{ijk})$). This computation is repeated for every process model to derive a list of recommended process models as shown in Table 23.

Table 22: Computing RATING for Waterfall process

Cost Profile =

$$\begin{bmatrix} C_1 & r_{1j} \times & (s_{1j1} \times a_{1j1}) & (s_{1j2} \times a_{1j2}) & (s_{1j3} \times a_{1j3}) & & t_{1j} \\ c_{11} & 3 & 1 & 0 & 0 & = & 3 \\ c_{12} & 3 & 0 & 1 & 0 & = & 3 \\ c_{13} & 0 & 0 & 0 & & = & 0 \\ c_{14} & 2 & 0 & 0 & 1 & = & 2 \\ c_{15} & 2 & 0 & 1 & 0 & = & 2 \\ c_{16} & 3 & 0 & 0 & 0 & = & 0 \\ c_{17} & 3 & -1 & 0 & 0 & = & -3 \\ c_{18} & 3 & 1 & 0 & 0 & = & 3 \\ c_{19} & 1 & 0 & 1 & 0 & = & 1 \end{bmatrix}$$

Time Profile =

$$\begin{bmatrix} C_2 & r_{2j} \times & (s_{2j1} \times a_{2j1}) & (s_{2j2} \times a_{2j2}) & (s_{2j3} \times a_{2j3}) & (s_{2j4} \times a_{2j4}) & & t_{2j} \\ c_{21} & 3 & 0 & 1 & 0 & & = & 3 \\ c_{22} & 2 & 0 & 0 & 1 & & = & 2 \\ c_{23} & 2 & 0 & 0 & 0 & & = & 0 \\ c_{24} & 2 & 1 & 0 & 0 & & = & 2 \\ c_{25} & 3 & 1 & 0 & 0 & & = & 3 \\ c_{26} & 2 & 0 & 1 & 0 & & = & 2 \\ c_{27} & 2 & 0 & 1 & 0 & & = & 2 \\ c_{28} & 0 & 0 & 0 & 1 & & = & 0 \\ c_{29} & 2 & 0 & 0 & 0 & 1 & = & 2 \\ c_{210} & 2 & 0 & 0 & 0 & & = & 0 \end{bmatrix}$$

Team Profile =

$$\begin{bmatrix} C_3 & r_{3j} \times & (s_{3j1} \times a_{3j1}) & (s_{3j2} \times a_{3j2}) & (s_{3j3} \times a_{3j3}) & & t_{3j} \\ c_{31} & 3 & 1 & 0 & 0 & = & 3 \\ c_{32} & 1 & 1 & 0 & 0 & = & 1 \\ c_{33} & 3 & 0 & 0 & 0 & = & 0 \\ c_{34} & 1 & 0 & 0 & 0 & = & 0 \\ c_{35} & 3 & 0 & 0 & 1 & = & 3 \end{bmatrix}$$

Development Profile =

$$\begin{bmatrix} C_4 & r_{4j} \times & (s_{4j1} \times a_{4j1}) & (s_{4j2} \times a_{4j2}) & (s_{4j3} \times a_{4j3}) & & t_{4j} \\ c_{41} & 2 & 0 & 1 & 0 & = & 2 \\ c_{42} & 2 & 0 & 1 & 0 & = & 2 \\ c_{43} & 3 & 0 & 1 & 0 & = & 3 \\ c_{44} & 2 & 1 & 0 & 0 & = & 2 \\ c_{45} & 2 & 0 & 0 & 1 & = & 2 \\ c_{46} & 2 & 1 & 0 & 0 & = & 2 \\ c_{47} & 2 & 0 & 1 & 0 & = & 2 \\ c_{48} & 2 & 0 & 0 & 0 & = & 0 \end{bmatrix}$$

Problem Profile =

$$\begin{bmatrix} C_5 & r_{5j} \times & (s_{5j1} \times a_{5j1}) & (s_{5j2} \times a_{5j2}) & (s_{5j3} \times a_{5j3}) & (s_{5j4} \times a_{5j4}) & & t_{5j} \\ c_{51} & 1 & 0 & 1 & 0 & & = & 1 \\ c_{52} & 2 & 0 & 0 & 1 & & = & 2 \\ c_{53} & 2 & 0 & 0 & 0 & 1 & = & 2 \\ c_{54} & 0 & 1 & 0 & 0 & & = & 0 \\ c_{55} & 2 & 0 & 1 & 0 & & = & 2 \\ c_{56} & 2 & 0 & 1 & 0 & & = & 2 \\ c_{57} & 3 & 0 & 0 & 0 & & = & 0 \\ c_{58} & 3 & 1 & 0 & 0 & & = & 3 \\ c_{59} & 3 & 0 & 0 & 0 & & = & 0 \\ c_{510} & 3 & 0 & 0 & 0 & & = & 0 \end{bmatrix}$$

Defect Profile =

$$\begin{bmatrix} C_6 & r_{6j} \times & (s_{6j1} \times a_{6j1}) & (s_{6j2} \times a_{6j2}) & (s_{6j3} \times a_{6j3}) & & t_{6j} \\ c_{61} & 3 & 0 & 0 & 1 & = & 3 \\ c_{62} & 3 & 0 & 0 & 0 & = & 0 \\ c_{63} & 2 & 0 & 1 & 0 & = & 2 \\ c_{64} & 2 & 0 & 1 & 0 & = & 2 \\ c_{65} & 2 & 0 & 0 & 1 & = & 2 \end{bmatrix}$$

Market/User Profile =

$$\begin{bmatrix} C_7 & r_{7j} \times & (s_{7j1} \times a_{7j1}) & (s_{7j2} \times a_{7j2}) & (s_{7j3} \times a_{7j3}) & & t_{7j} \\ c_{71} & 3 & 1 & 0 & 0 & = & 3 \\ c_{72} & 3 & 0 & -1 & 0 & = & -3 \\ c_{73} & 2 & 0 & 0 & 1 & = & 2 \\ c_{74} & 3 & 0 & 0 & 1 & = & 3 \\ c_{75} & 3 & 0 & 0 & 1 & = & 3 \\ c_{76} & 1 & 1 & 0 & 0 & = & 1 \\ c_{77} & 3 & 1 & 0 & 0 & = & 3 \\ c_{78} & 3 & 0 & 0 & -1 & = & -3 \\ c_{79} & 1 & 0 & 0 & 0 & = & 0 \\ c_{710} & 3 & 1 & 0 & 0 & = & 3 \\ c_{711} & 3 & 1 & 0 & 0 & = & 3 \end{bmatrix}$$

Management Profile =

$$\begin{bmatrix} C_8 & r_{8j} \times & (s_{8j1} \times a_{8j1}) & (s_{8j2} \times a_{8j2}) & (s_{8j3} \times a_{8j3}) & & t_{8j} \\ c_{81} & 3 & 0 & 0 & 1 & = & 3 \\ c_{82} & 3 & 0 & 0 & 1 & = & 3 \\ c_{83} & 3 & 0 & 0 & 1 & = & 3 \\ c_{84} & 2 & 0 & 1 & 0 & = & 2 \\ c_{85} & 3 & 0 & 0 & 1 & = & 3 \\ c_{86} & 2 & 0 & 1 & 0 & = & 2 \end{bmatrix}$$

Table 23 displays the rating for the subset of development lifecycles considered for this study. The process with the highest rating is considered the most suitable as it covers more of the project goals than the rest. For Project A (where the cumulative project characteristics scored up to 145), Spiral model (RATING = 133) matched most of the project requirements, closely followed by Waterfall model (RATING = 101) and the Feature Driven Development (RATING = 100). XP (RATING = 46) and Scrum (RATING = 54) scored much less as they are more applicable for projects with shorter release dates, constantly changing requirements and relatively low risks. Project A is best handled by processes that handle higher risks, stable requirements and heavy documentation like the Spiral or the Waterfall model.

Table 23: Ratings for all the process models

Process Model	RATING (Project A characteristics score up to 145)
Spiral	133
Waterfall	101
FDD	100
Scrum	54
XP	46

5. *Analyze $(s_{ijk} \times a_{ijk})$'s of each profile, p_i, for the selected process model to determine the key areas for improvement* –The $(s_{ijk} \times a_{ijk})$ values represent

where the process meets the project needs, where it is not appropriate and where the process can negatively impact project development. The criteria that are not satisfied by the process and/or negatively influenced by the use of the recommended process are considered major areas for improvement. For this study, graphs are used to plot the $t_{ij} = r_{ij} \times \sum_k (s_{ijk} \times a_{ijk})$ values for the process and the r_{ij} values for the project for each criterion, c_{ij}, to identify project attributes that are not covered by the process, i.e. the areas where the process diverges from the project goals. Figures 12 – 19 depict the comparison graphs for the different profiles.

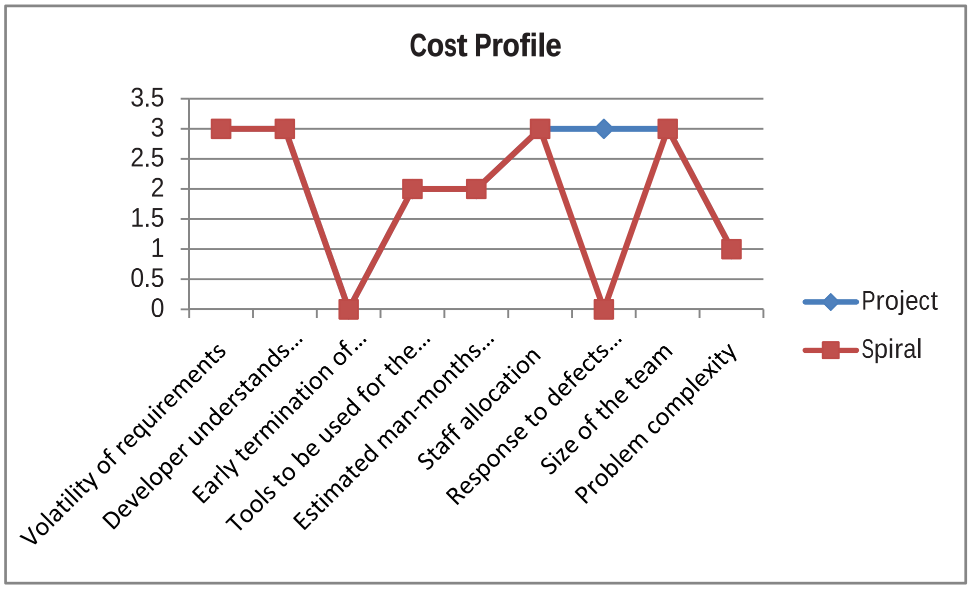


Figure 12: Cost profile analysis

As can be seen in the Figure 12, Spiral modeling meets most of the project goals except for ‘Response to defects found during operation’. Spiral modeling considers the maintenance phase of the project as part of the Spiral and feeds customer complaints or validation data for the next spiral’s requirements cycle. As a medical device company, critical defects in the field need to be handled in parallel to software development for Project A. The Spiral model can easily be adjusted to create internal spiral spin-offs where a part of the development team is assigned to analyze the risks of resolving the defects within this release or demoting them for the next spiral.

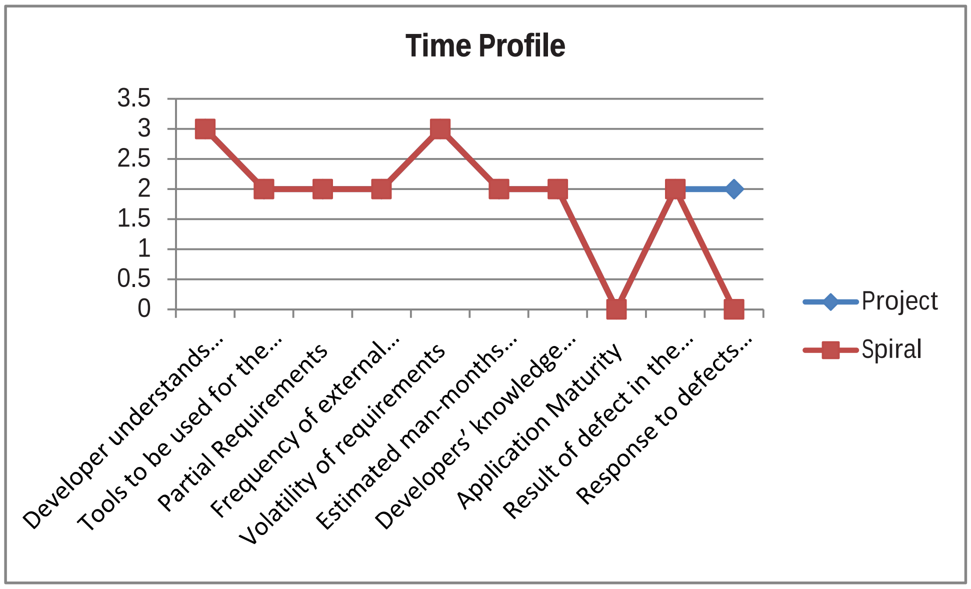


Figure 13: Time profile analysis

Similar to the cost profile, the 'Response to defects during operation' criterion affects the time profile as well and needs to be adjusted to be handled in parallel to software development.

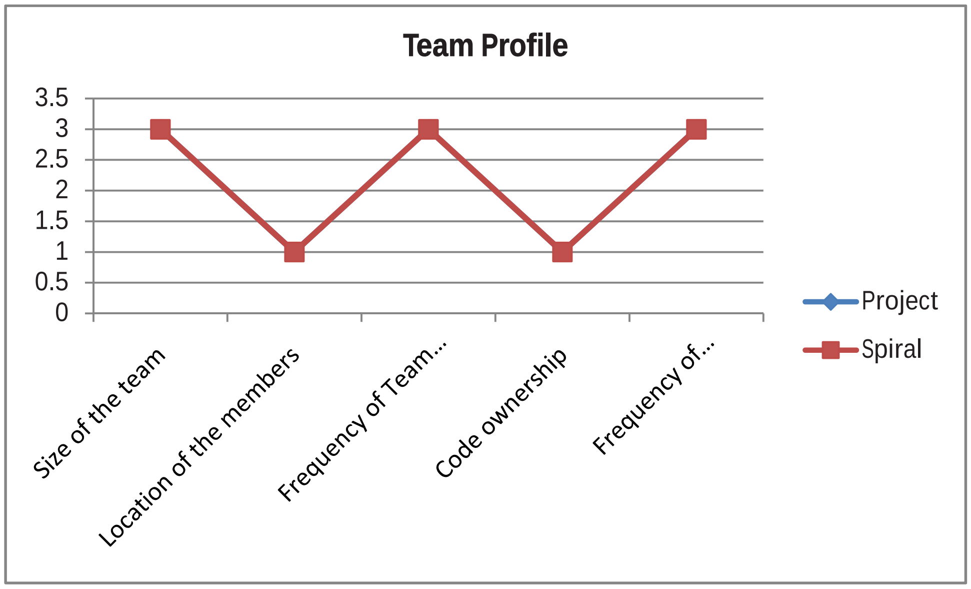


Figure 14: Team profile analysis

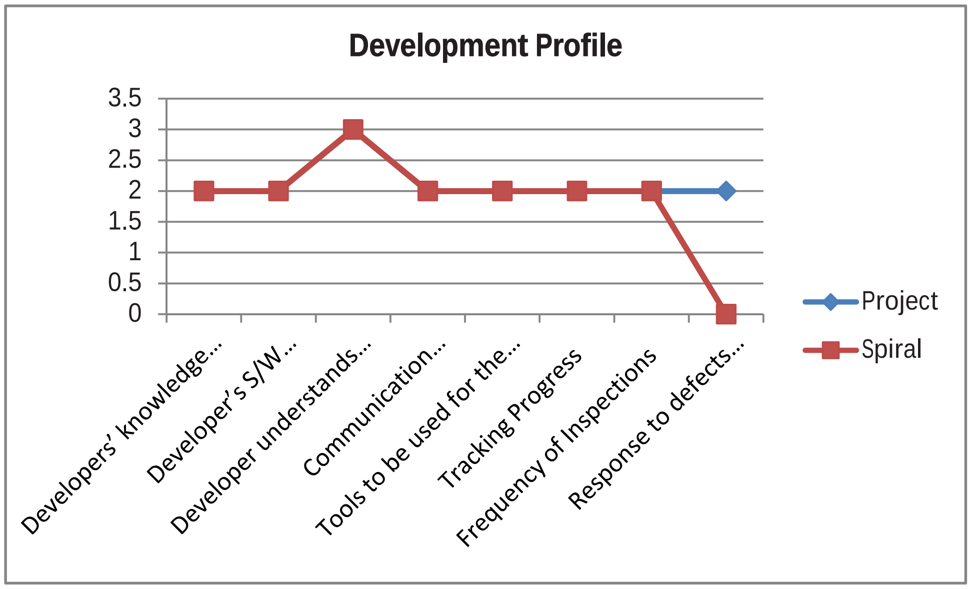


Figure 15: Development profile analysis

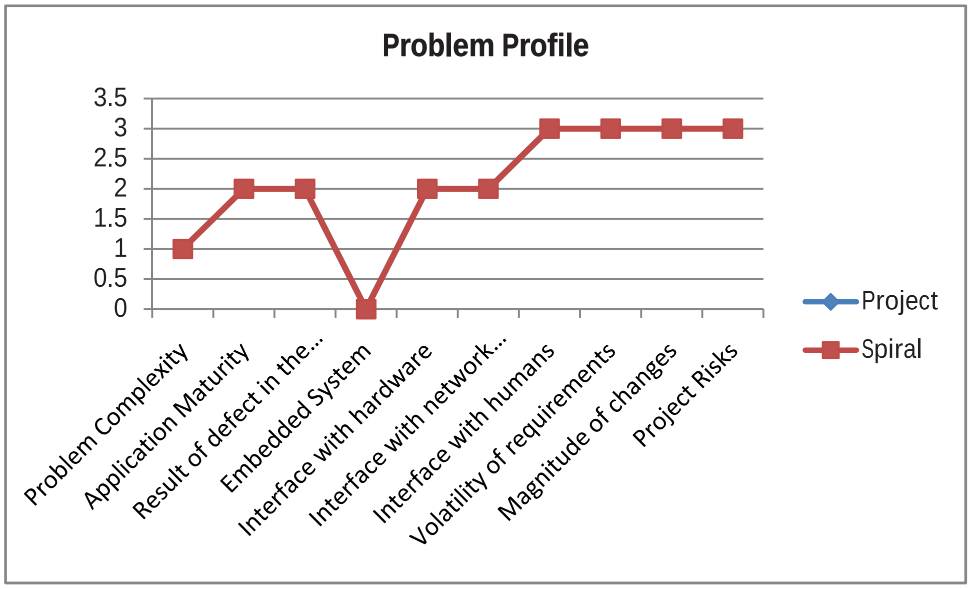


Figure 16: Problem profile analysis

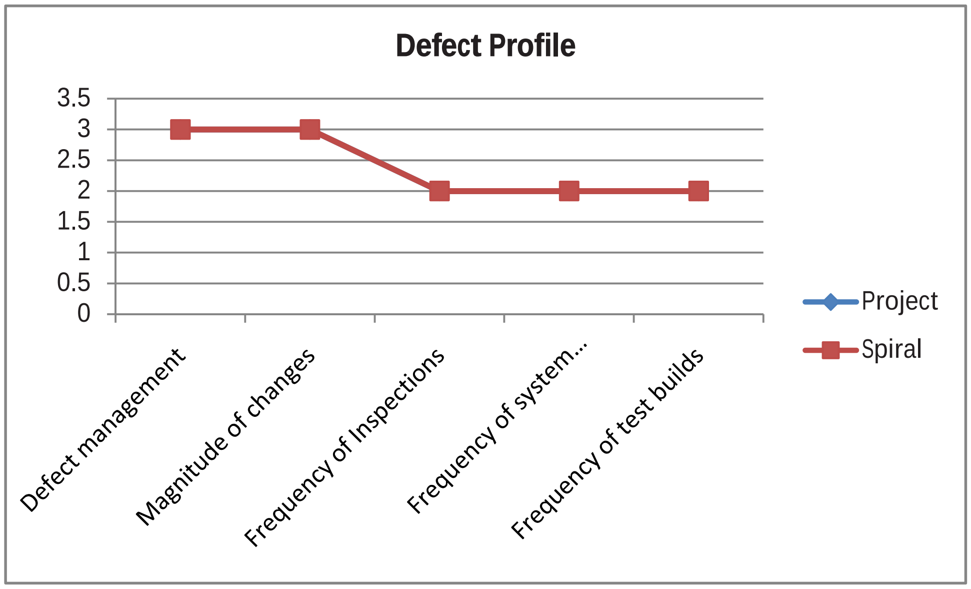


Figure 17: Defect profile analysis

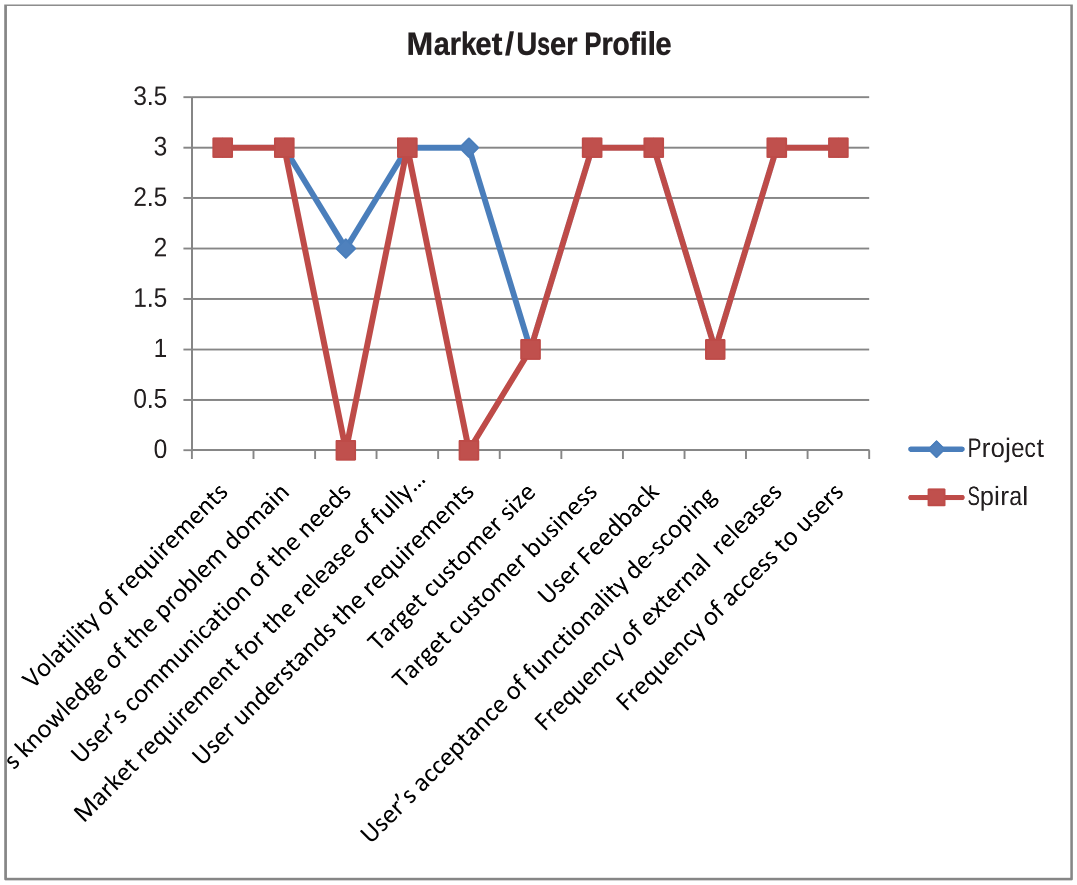


Figure 18: Market/User profile analysis

As shown in Figure 18, the Spiral model is not applicable to expressive users and users with excellent understanding of the requirements. The following explains the reasons behind each:

User's communication of the user needs – Spiral development uses proof of concept prototypes for conceptualizing design ideas and for gaining user feedbacks on the proposed solution. If the users are expressive about the needs, developing prototypes to gain user feedback on design ideas will increase the cost of development along with delaying the schedule of the project.

User understands the requirements – The proof of concept prototype also assists in making sure the user understands the solution domain as well as the problem domain. Creating prototypes for users who are familiar with both is a waste of time for the development team and that time should be reallocated effectively.

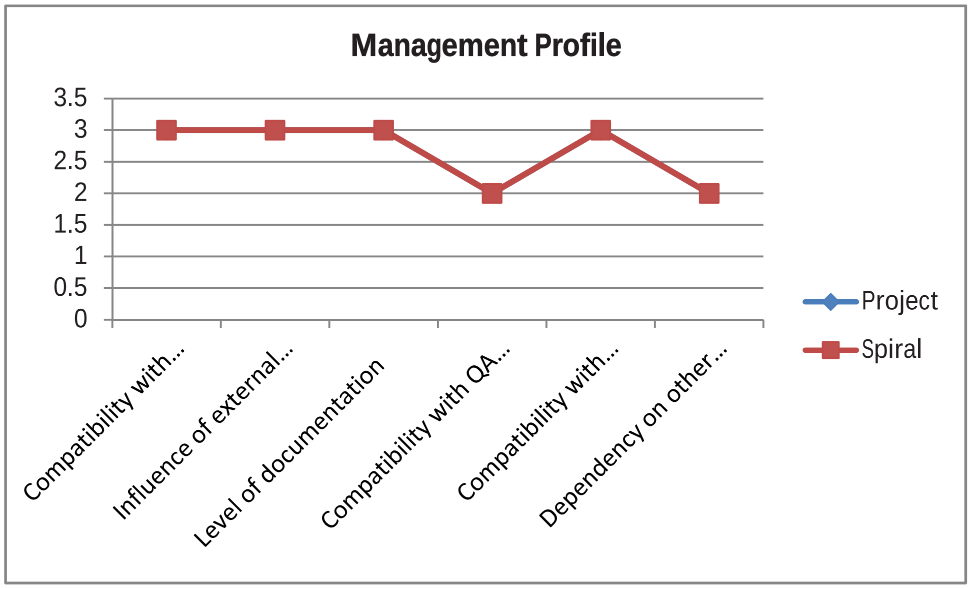


Figure 19: Management profile analysis

6. *Suggest improvements by evaluating* $(s_{ijk} \times a_{ijk})$*'s for other processes* – Figures 20 – 21 depict how Waterfall and FDD resolve the project's market/user needs. Both Waterfall process and FDD are applicable to projects with expressive users and users who understand the requirements very well. However, both fail to address other major project goals. Strict Waterfall depends on the users' knowledge of the problem domain and their expression of the requirements. Vague requirements and users' lack of communication can negatively impact the development process. The Waterfall model uses customer input as a written contract of what is required in the system while

using preliminary design to scope out ideas. Customer interaction is minimal during this period. FDD uses customer and developer teams for creating an object model of the domain, therefore it is essential that the customers are aware of their needs and can express them clearly. XP and Scrum are generally not applicable for projects with not much customer interaction and where project requirements are known. In order to improve the Spiral modeling for Project A, the Waterfall approach of requirements specifications and preliminary design can be considered over rapid prototypes. As the object model for the project is already developed, FDD would provide very little benefit.

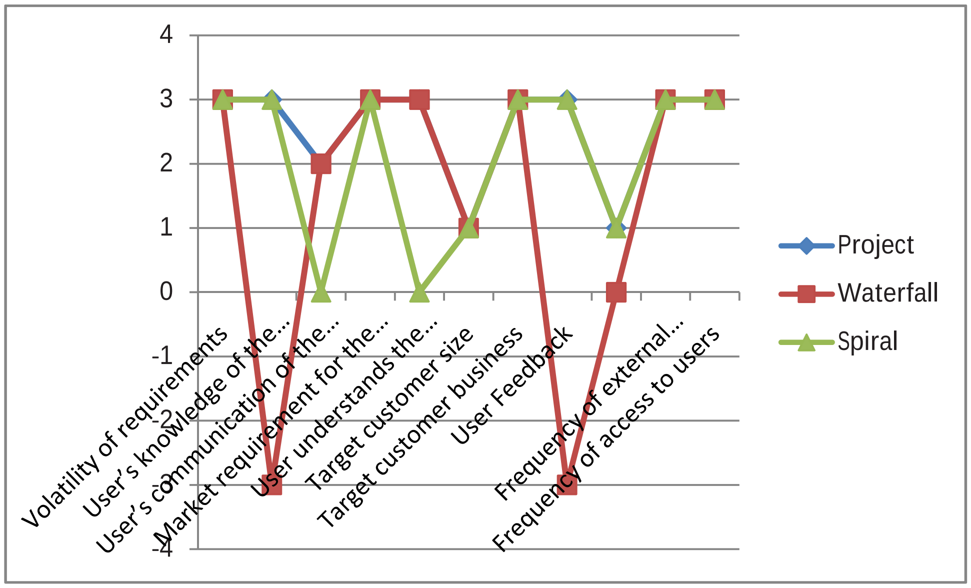


Figure 20: Spiral model compared with Waterfall for Market/User profile

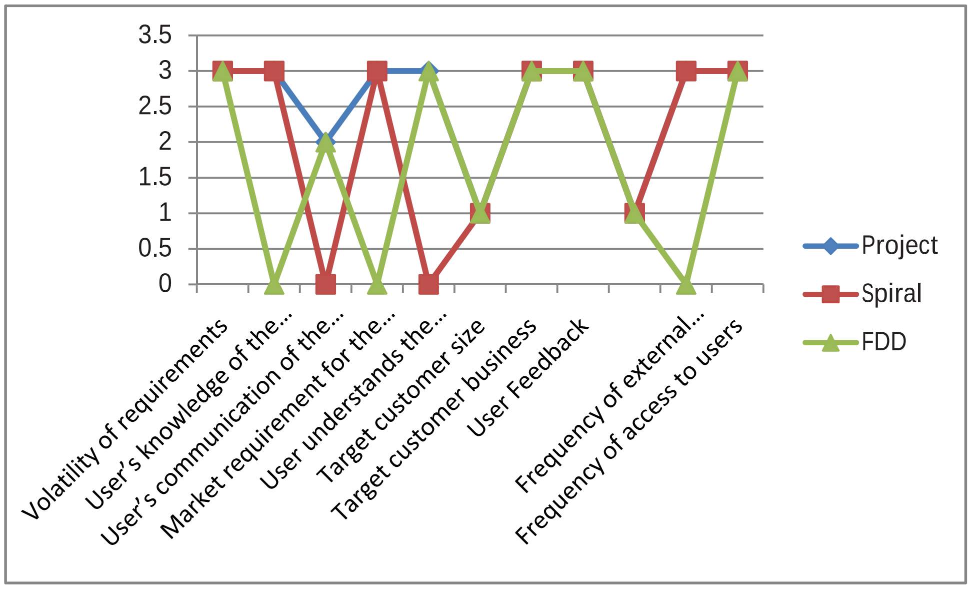


Figure 21: Spiral model compared with FDD for Market/User profile

7. *Develop metrics to analyze the improvement with respect to the profile* – As described by [Basili, 1985], metrics can be either objective or subjective. Subjective metrics like rating the new process improvements with respect to the old process can be evaluated by interviewing the developers at the end of the project. Objective metrics are absolute measures and can be evaluated through data collection over the development period. In order to assess the process improvements made for this project both subjective and objective metrics can be used as shown in the following table.

Table 24: Improvement metrics for process improvement evaluation

Improvement	Type of metric	Metrics
Inner spiral for addressing defects	Objective	▪ number of defects found during operation ▪ time taken to resolve the defects ▪ ratio of developer allocation for each defect ▪ number of defects going into the current release ▪ number of defects going demoted to next spiral
	Subjective	▪ Was there a clear division between defects to be demoted and defects to be resolved?

		▪ How did addressing the defects affect current development?
Software requirements specification and design prototypes in place of user prototypes	Objective	▪ Time spent on developing preliminary design prototypes ▪ Number of prototypes created
	Subjective	▪ Level of customer satisfaction after product development ▪ Did the decrease in the number of prototypes affect the understanding of the project

8. *Analyze and select tools that address the improvement goals* – A proper defect management system can address most defect related data collection metrics. A scheduling and resource management system can be used to measure the development times and staff allocation. As Project A has a set of established tools, it is important to analyze the current set of tools against the improvement goals and metrics to determine if they can be used to collect data. Adoption of new tools in an established environment has a significant effect on the cost of the project due to necessary insertion methods like personnel training etc. The benefit of adding a new tool needs to be tallied with the amount of effort and cost it would take to integrate it into the development environment.
9. *Apply the process and collect data for the metrics* – The next step in the process is to exercise the selected process and collect the data for the metrics defined. Recording this data in a corporate database provides project managers with the means to evaluate the process improvement goals from their project with past attempts [Basili, 1985]. Objective metrics can be measured using automatic data collection performed by tools identified in the previous step. Subjective metrics can be quantified by surveying the

developers, customers, QA etc., and performing post-mortem interviews to evaluate the performance of the implemented improvements.

10. *Analyze the metrics to determine areas for improvement in future versions of the project* – Data collected from previous product development and process improvement efforts can be used to identify areas of improvements for the next software development. Specifically, failed attempts can be profiled using UPSM and used to tailor the process to the new goals of the project.

5.4 Analysis

The UPSM approach was applied to Project A’s attributes using the profiles and criteria defined in Section 5.1.2. It was concluded that the Spiral model is best suited for Project A’s goals and characteristics followed by the Waterfall and Feature Driven Development models as the project requirements are relatively stable, the project has high risks, is fairly complex and is in maintenance. Improvement objectives were determined and the process was tailored to address the improvement goals. The results derived by the UPSM framework are very close to how the project is being handled currently. When new requirements come in due to changes in the market or to the embedded device the software interfaces with, the risks are analyzed and proof-of-concept prototypes or “spikes” are created to establish the solution. User feedback is rarely acquired until the end of the project when the “to-be released” version is validated with the users. Process tailoring defined using the UPSM follows a similar philosophy. The process was tailored to incorporate the Waterfall model’s preliminary design approach. Proof-of-concept prototypes were recommended to be developed to gain insight into preliminary design of the system and not to gain user feedback. This reduces the time spent on developing prototypes when the user is already well aware of the solution domain and all the requirements are known. The process was also tailored to handle defects from the field using internal spirals, whereas the real project team uses a review board to analyze the risks of incorporating the defects in the release and proceeds with resolving the defects in parallel to project development. The development process and the organizational requirements are well established for Project A and implementation of a new process might be very difficult for several reasons. Applying UPSM to select an appropriate process for Project A might be inefficient as the process is standardized and the team will be reluctant to adapt to any new processes. The cost of applying a new process to such an environment is also high and management may not be willing to invest in it. However, the UPSM can be used to determine improvements for the established process and suggest strategies to execute the improvements.

As the project evolves and grows, the goals and attributes of the project might change. The UPSM framework provides means to address these changes through continuous improvement and tailoring. For example, sometime in the future, Project A’s customers may require quick additional functionalities and are willing to collaborate. Spiral development may still be the best approach; however, now the process needs to accommodate the new characteristics and must be tailored for the project. The UPSM can be utilized to identify areas of improvement and incorporate practices from other methodologies. Creating performance metrics allows the managers to monitor the effect of the new improvement strategies and quantifiably determine whether the actual benefits of the process improvements measure up to the expected benefits. The improvement goals of the project also form the selection criteria for tool selection and insertion. Although, the UPSM process identifies areas for improvement and provides options for tool selection and process advancement, the ultimate decision to proceed with the changes lies with management. The benefits of the implementing the recommendations must be weighed against the effort, cost and the time that will be put into using a new process, tool or improvement.

5.5 Comparison with Past Research

A common theme visible in related studies performed in the past is the evaluation of software processes from a single point of view. As mentioned in Section 3.6, although such approach narrows the selection criteria it fails to address issues encountered in other areas. The UPSM endeavors to rectify this situation by establishing project profiles for evaluation purposes. The profiles abstract information regarding different management concerns, like cost, schedule, resources etc., into a concrete view point which is used to assess the project and processes from the same angle. Such abstraction of project characteristics allows managers to focus on scouting improvement areas and establishing improvement goals to address the management concern. In contrast to the single point of view evaluation, project profiles allow evaluation to be performed from different angles and collectively determine the most suitable process for the project.

The UPSM framework is based on [Alexander & Davis, 1991]'s criteria based selection model. Alexander et al. define a set of criteria from various aspects of project development and management to judge various processes and project characteristics and match the two together. The UPSM model extends this approach by addressing some assumptions that Alexander et al. make in their evaluation framework. Alexander et al. assume that each criterion has the same priority from the management perspective. This assumption is not very realistic as it fails to accommodate projects that emphasis quick deliveries more than tool integration in the project, for example. The UPSM attempts to resolve this issue by rating the criterion based on their importance to the project with respect to the profile. This rating is used to influence the process selections such that the processes addressing the most important criterion are placed higher in the suitable processes list.

In their concluding statements, [Alexander & Davis, 1991] mention that "negative" values for criteria with negative influence "could prompt plans to avoid problems when the recommended process model is not used or to tailor a process model to fit all of the evaluations [p528]." The UPSM incorporates this idea by evaluating values that have a negative influence on the project profile with a -1. For example, projects with heavy documentation needs for management standards suffer when using Extreme Programming. When the process matchup ratings are computed, the negative rating can be used to identify areas in the software lifecycle that harm the project when the recommended process is applied. This evaluation is also used to identify areas in the process that can benefit from tailoring them with elements from other processes that meet the project goals.

The UPSM also attempts to integrate [Basili, 1985]'s goal-question-metric theory to assist managers in continuously improving the selected process to satisfy the project's current goals. Although Basili suggests the development of goals based on the manager's experience and knowledge of the project, for more novice managers the goals can be misrepresented and can potentially lead the system in an incorrect direction. The UPSM established the improvement goals from the criteria that fail to address the project needs. These goals are concrete and can be used to develop metrics for assessing the process and tools performance over the project life cycle.

The UPSM criteria and profiles are developed by incorporating various viewpoints introduced by related literature (e.g. user characteristics defined by [Davis, Bersoff & Comer, 1988] and problem characteristics identified in [Kettunen & Laanti, 2004]) and information gathered from the interviews conducted. The criteria and profiles form the backbone of the systematic approach to determining the most suitable process for the project which is absent in [Davis, Bersoff & Comer, 1988] and [Kettunen & Laanti, 2004]. The primary management concerns identified by Davis et al. form the basis for the profile subjects.

Overall, the UPSM attempts to integrate the strengths of the past work in this area to present a systematic and formalized framework for matching project characteristics to process characteristics that address different management concerns. It also tries to resolve some of the weaknesses seen in past work. A key difference between the related studies made in the past and the UPSM framework is that the latter facilitates the identification and successive correction of potential risk areas in the process when applied to a particular project. The UPSM tailoring strategy is different from [Basili & Rombach, 1987]'s defect schemes in the sense it suggests creation of hybrid processes with elements from other processes in addition to past and current performance analysis of improvement goals. Data collected from interviews with participants from different software projects suggest that project teams tend to mix and match various processes to create a suitable process for the project. The UPSM facilitates this understanding by identifying the weakest areas of the recommended processes and determining which elements from other processes best match the project goals in those areas. The decision to make the change in the current project and estimation of the costs lie with the project manager, the UPSM assists in identifying options for making the change.

CHAPTER 6
Future Work

The general approach adopted for this thesis was to gather qualitative information about projects and their processes by conducting face-to-face interviews with representatives from a range of project teams in various application domains. Although, structured and semi-structured interviews provide a wealth of insight into process executions, the conclusions drawn are limited by the size of the population interviewed. Statistical surveys about the problems encountered in project development and maintenance, development life cycles employed, and project characteristics can be administered to a larger sample spanning many teams, problem domains and software processes. Quantitative data collected from these surveys can be used to validate the qualitative information provided by the interviews and support the conclusions drawn. Surveys cover more ground than face-to-face interviews and can standardize the data collected to be presented in a statistical fashion. In future studies, surveys or questionnaires can be used to investigate a larger population and build a knowledge base about the projects, processes, teams and tools. A subset of the population can then be selected to be interviewed based on the data collected and the interviews can focus more on management decisions taken to improve the quality and productivity of the development process.

The scope of this thesis involved formalizing a process selection model that evaluates a project from different viewpoints and illustrating its application on a project. However, in order to evaluate its effectiveness, the recommended process should be deployed and monitored throughout the development period. [Basili, 1985]'s objective and subjective metrics can be used to facilitate the evaluation. Objective metrics like the total development time, number of defects produced, number of defects addressed, rate of development cost increase, etc. can be measured throughout the project development to determine how the new process or adjustments made to the process hold up in the project environment. Subjective metrics in the form of post-mortem interviews or questionnaires can be used to evaluate how close the process came to addressing the project's needs. Tools can be used to un-intrusively collect data throughout the development life cycle and automate the evaluation process. Although, the UPSM defines a mechanism to select an appropriate process for a given project, its reliability can only be assessed from executing the recommended process and evaluating its performance with respect to the project's goals and environment.

To focus the thesis on the process selection framework, a subset of modern development life cycles was used. In reality a number of software processes exist and are exercised by many industries. Evaluating each process individually as is proposed by the UPSM might be very time consuming. Tools can be used to automate the whole process by persisting applicability matrices in a back-end database and generating graphs and evaluation reports suggesting improvements. They can also be designed to un-intrusively measure the performance of a new process or improvements to a process and generate reports assessing their effectiveness. Another way to support multiple life cycles in the framework is to apply [Alexander & Davis, 1991]'s process hierarchy approach. General trends and patterns can be abstracted from different processes to be evaluated against a project's needs. For example, Scrum and XP employ iterative development with tight timeboxes for each release, while the Waterfall and Spiral models execute a phased development. Using Alexander et al.'s process hierarchy approach, process trends or patterns can be evaluated against the project characteristics first. This will narrow the selection to the processes that display the trend. The subset of processes identified can then be subjected to the UPSM framework to determine the most suitable life cycle for the project.

Estimation of project characteristics with Participant A using the profiles and criteria provided useful insight into what values were ambiguous and which were not really applicable to the profile from a management viewpoint. Adjustments were made to the profile and various criteria from the input received. Future studies should perform walkthroughs of the profiles and criteria with more companies and participants to standardize the UPSM structure and eliminate areas of ambiguity and confusion in the criteria selection.

Overall, there are several directions this research can take to improve its approach and process selection design. This section suggests some improvements to the research methodology and the proposed model and leaves room for future studies to provide a more robust and systematic approach for process selection.

CHAPTER 7
Conclusion

There are several development life cycles and methodologies in existence today, ranging from the Waterfall process described by Royce [1987] to the latest Agile techniques [Manifesto, 2001]. It is not always apparent which process is best suited for the needs of a project, and more often than not, such decisions are based on organizational standards or decided through ad-hoc means. Over time, researchers have proposed various models to formalize the selection of a suitable process for a project, each with their advantages and disadvantages. The Unified Process Selection Model presented in this thesis attempts to address the weaknesses of past research in this area while incorporating their strengths to define a systematic and formalized approach to selecting an appropriate process for a project and tailoring it to a project’s needs.

The strengths of the UPSM lie in being able to analyze a given project from different viewpoints and identify areas for improvement. Project managers determine what characteristics best define their project and rate them based on how important it is to the project’s cost, schedule, etc. The ratings and project attributes are used to decide which of a given set of development processes best suit a project’s needs. “Negative” values assigned to project characteristics assist in identifying areas where the recommended process can potentially harm the project. The UPSM attempts to resolve these harmful areas by fusing “positive” aspects of other processes with the recommended life cycle to create a hybrid process unique to the project’s goals and by identifying tools to facilitate the fusion. Furthermore, the tools and improvements are continuously monitored to determine their effectiveness for the project and to establish improvement goals for future software development.

In order to demonstrate the capabilities of this framework, the UPSM was applied to a real software project. Evaluation of the project's attributes concluded that the spiral model was best suited for it and a number of improvement goals were identified to address those characteristics that were inefficiently handled by the spiral model. The process was combined with the traditional waterfall methodology to resolve these issues and to create a hybrid spiral-waterfall methodology specific to the project's needs. This example illustrated how the UPSM approach can be used to select and tailor an appropriate process for the project. The results closely reflected how the project is being handled in reality and also provided the means to improve future versions of the software through continuous data collection and performance evaluation.

Although selecting the right development methodology and tailoring it to a project's goals and needs increases the efficiency and productivity of a project, it is not the sole contributor to a project's success. Other factors like a project's CMM level and the developer's experience play a role in defining the failure and success of a project. As described by [Highsmith, 2002], "a project with talented people and a streamlined process has a high probability of success, while a project staged with under-talented people—no matter how good their process—will probably fail [p271]." Nonetheless, establishing a development process tailored for a project's needs effectively reduces the cost of the project and satisfies the needs of the customer in a timely fashion.

References

Alexander, L. C., & Davis, A. M. (1991). Criteria for selecting software process models. *Proceedings of the Fifteenth Annual International* (pp. 521-528). Tokyo, Japan: Computer Software and Applications Conference.

Basili, V. (1985). *Quantitative Evaluation of Software Methodology*. Maryland: University of Maryland.

Basili, V. R., & Rombach, D. H. (1987). Tailoring the software process to project goals and environments. In *ICSE '87: Proceedings of the 9th international conference on Software Engineering* (pp. 345-357). Monterey, California, United States: IEEE Computer Society Press.

Beck, K. (1999). Embracing change with extreme programming. *Computer , 32* (10), 70-77.

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

Brown, A. W., Carney, D. J., Morris, E. J., Smith, D. B., & Zarrella, P. F. (1994). *Principles of CASE Tool Integration.* New York: Oxford University Press.

Brown, W., McCormick III, H. W., & Thomas, S. (2000). *AntiPatterns in Project Management.* Wiley.

Bruckhaus, T. (1993). The impact of inserting a tool into a software process. *CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research* (pp. 250-264). Toronto, Ontario, Canada: IBM Press.

Bruckhaus, T. (1994). TIM: a tool insertion method. *CASCON '94: Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research* (pp. 1-13). Toronto, Ontario, Canada: IBM Press.

C3Team. (1998, October). Chrysler Goes to "Extremes". *Distributed Object Computing* , 24-28.

Cockburn, A. (2000). Selecting a Project 's Methodology. *IEEE Software , 17* (4), 64-71.

Davis, A. (1994). Fifteen principles of software engineering. *Software, IEEE , 11* (6), 94-96, 101.

Davis, A. M. (1990). *Software Requirements: Analysis and Specification.* Englewood Cliffs, New Jersey: Prentice-Hall.

Davis, A., Bersoff, E., & Comer, E. (1988). A Strategy for Comparing Alternative Software Development Life Cycle Models. *IEEE Transactions on Software Engineering , 14* (10), 1453-1461.

Ewusi-Mensah, K. (2003). *Software Development Failures: Anatomy of Abandoned Failures.* Cambridge, Massachusettes: MIT Press.

GSAM. (2003). *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems.* Utah.

Highsmith, J. (2002). *Agile Software Development Ecosystems.* Addison-Wesley.

Jalote, P. (2005). *An Integrated Approach to Software Engineering* (3rd ed.). Springer.

Jalote, P. (2002). *Software Project Management in Practice.* Addison-Wesley.

Jurison, J. (1999). Software project management: the manager's view. *Communications of the AIS , 2* (17).

Kettunen, P., & Laanti, M. (2004). How to steer an embedded software project: tactics for selecting the software process model. *Information and Software Technology , 47* (9), 587-608.

Lewis, T. G. (1991). *CASE: Computer-Aided Software Engineering.* New York: Van Nostrand Reinhold.

MacCormack, A., Kemerer, C. F., Cusumano, M. A., & Crandall, B. (2003). Trade-offs between Productivity and Quality in Selecting Software Development Practices. *IEEE Software , 20* (5), 78-85.

Martin, R. C. (2003). *Agile Software Development: Principles, Patterns, and Practices.* NJ: Prentice Hall.

Mayhew, D. J. (1999). User Profiles. In *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design (Interactive Technologies)* (pp. 35-66). Morgan Kaufmann.

Mcclure, C. (1989). *CASE Is Software Automation.* Englewood Cliffs, New Jersey: Prentice Hall.

Nawrocki, J., Jasinski, M., Walter, B., & Wojciechowski, A. (2002). Extreme programming modified: embrace requirements engineering practices. *Proceedings of*

IEEE Joint International Conference on Requirements Engineering, 2002. (pp. 303-310). IEEE Press.

Palmer, S. R., & Felsing, J. M. (2002). *A Practical Guide to Feature-Driven Development.* NJ: Prentice Hall Inc.

Prather, B. (1993). Critical failure points of CASE tool evaluation and selection. *Computer-Aided Software Engineering, 1993. CASE '93.* (pp. 60-63). Singapore: Proceeding of the Sixth International Workshop.

Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. In *ICSE '87: Proceedings of the 9th international conference on Software Engineering* (pp. 328-338). Monterey, California, United States: IEEE Computer Society Press.

Schach, S. R. (2007). *Object-Oriented & Classical Software Engineering* (7th ed.). McGraw-Hill.

Simon, A. R. (1993). *The Integrated CASE Tools Handbook.* New York: Van Nostrand Reinhold.

Succi, G., & Marchesi, M. (2001). *Extreme Programming Examined* . Addison-Wesley.

Sutherland, J., & Schwaber, K. (2007). *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process.* Boston: Scrum, Inc.

The Agile Manifesto. (2001). Retrieved from agilemanifesto.org

Appendix A. Applicability Matrices

A.1 Waterfall Method

Cost Profile =

$$\begin{bmatrix} Criteria_1 & v_{1j1} & v_{1j2} & v_{1j3} \\ c_{11} & 1 & -1 & -1 \\ c_{12} & -1 & 1 & 1 \\ c_{13} & 0 & 0 & \\ c_{14} & -1 & 1 & 1 \\ c_{15} & 0 & 1 & 1 \\ c_{16} & -1 & 0 & 1 \\ c_{17} & -1 & 0 & 0 \\ c_{18} & 1 & 1 & 1 \\ c_{19} & 0 & 1 & 1 \end{bmatrix}$$

Time Profile =

$$\begin{bmatrix} Criteria_2 & v_{2j1} & v_{2j2} & v_{2j3} & v_{2j4} \\ c_{21} & -1 & 1 & 1 & \\ c_{22} & -1 & 1 & 1 & \\ c_{23} & 0 & -1 & -1 & \\ c_{24} & 1 & -1 & -1 & \\ c_{25} & 1 & -1 & -1 & \\ c_{26} & 0 & 1 & 1 & \\ c_{27} & -1 & 1 & 1 & \\ c_{28} & 0 & -1 & 1 & \\ c_{29} & 1 & 1 & 1 & 1 \\ c_{210} & -1 & 0 & 0 & \end{bmatrix}$$

Team Profile =

$$\begin{bmatrix} Criteria_3 & v_{3j1} & v_{3j2} & v_{3j3} \\ c_{31} & 1 & 1 & 1 \\ c_{32} & 1 & 1 & 1 \\ c_{33} & -1 & 1 & 0 \\ c_{34} & 0 & 0 & 0 \\ c_{35} & -1 & 1 & 1 \end{bmatrix}$$

Development Profile =

$$\begin{bmatrix} Criteria_4 & v_{4j1} & v_{4j2} & v_{4j3} \\ c_{41} & -1 & 1 & 1 \\ c_{42} & 1 & 1 & 1 \\ c_{43} & -1 & 1 & 1 \\ c_{44} & 1 & 1 & 0 \\ c_{45} & -1 & 1 & 1 \\ c_{46} & 1 & 0 & 0 \\ c_{47} & -1 & 1 & 0 \\ c_{48} & -1 & 0 & 0 \end{bmatrix}$$

Problem Profile =

$$\begin{bmatrix} Criteria_5 & v_{5j1} & v_{5j2} & v_{5j3} & v_{5j4} \\ c_{51} & 0 & 1 & 1 & \\ c_{52} & 0 & -1 & 1 & \\ c_{53} & 0 & 0 & 1 & 1 \\ c_{54} & 1 & 1 & 1 & \\ c_{55} & 1 & 1 & 1 & \\ c_{56} & 1 & 1 & 1 & \\ c_{57} & 1 & 1 & 0 & \\ c_{58} & 1 & -1 & -1 & \\ c_{59} & 0 & 0 & 0 & \\ c_{510} & 1 & 1 & 0 & \end{bmatrix}$$

Defect Profile =

$$\begin{bmatrix} Criteria_6 & v_{6j1} & v_{6j2} & v_{6j3} \\ c_{61} & 0 & 1 & 1 \\ c_{62} & 0 & 0 & 0 \\ c_{63} & -1 & 1 & 0 \\ c_{64} & 1 & 1 & 0 \\ c_{65} & 0 & 0 & 1 \end{bmatrix}$$

Market/User Profile =

$$\left[\begin{array}{cccc} Criteria_7 & v_{7j1} & v_{7j2} & v_{7j3} \\ c_{71} & 1 & -1 & -1 \\ c_{72} & -1 & -1 & 1 \\ c_{73} & -1 & -1 & 1 \\ c_{74} & 0 & 1 & 1 \\ c_{75} & -1 & -1 & 1 \\ c_{76} & 1 & 1 & 1 \\ c_{77} & 1 & 1 & 1 \\ c_{78} & 0 & 1 & -1 \\ c_{79} & 0 & 0 & 0 \\ c_{710} & 1 & -1 & -1 \\ c_{711} & 1 & 0 & 0 \end{array}\right]$$

Management Profile =

$$\left[\begin{array}{cccc} Criteria_8 & v_{8j1} & v_{8j2} & v_{8j3} \\ c_{81} & 1 & 1 & 1 \\ c_{82} & 1 & 1 & 1 \\ c_{83} & -1 & -1 & 1 \\ c_{84} & 1 & 1 & 1 \\ c_{85} & 1 & 1 & 1 \\ c_{86} & 1 & 1 & 1 \end{array}\right]$$

A.2 Spiral Model

Cost Profile =

$$\left[\begin{array}{cccc} Criteria_1 & v_{1j1} & v_{1j2} & v_{1j3} \\ c_{11} & 1 & 1 & 0 \\ c_{12} & 1 & 1 & 1 \\ c_{13} & 1 & 1 & \\ c_{14} & 0 & 1 & 1 \\ c_{15} & 1 & 1 & 1 \\ c_{16} & -1 & 1 & 1 \\ c_{17} & 0 & 0 & 1 \\ c_{18} & 1 & 1 & 1 \\ c_{19} & -1 & 1 & 1 \end{array}\right]$$

Time Profile =

$$\left[\begin{array}{ccccc} Criteria_2 & v_{2j1} & v_{2j2} & v_{2j3} & v_{2j4} \\ c_{21} & 1 & 1 & 1 & \\ c_{22} & 0 & 1 & 1 & \\ c_{23} & 1 & -1 & -1 & \\ c_{24} & 1 & 1 & -1 & \\ c_{25} & 1 & 1 & 0 & \\ c_{26} & 1 & 1 & 1 & \\ c_{27} & 0 & 1 & 1 & \\ c_{28} & 1 & -1 & 1 & \\ c_{29} & 0 & 0 & 1 & 1 \\ c_{210} & 0 & 0 & 1 & \end{array}\right]$$

Team Profile =

$$\left[\begin{array}{cccc} Criteria_3 & v_{3j1} & v_{3j2} & v_{3j3} \\ c_{31} & 1 & 1 & 1 \\ c_{32} & 1 & 1 & 1 \\ c_{33} & 0 & 1 & 1 \\ c_{34} & 1 & 0 & -1 \\ c_{35} & -1 & 1 & 1 \end{array}\right]$$

Development Profile =

$$\left[\begin{array}{cccc} Criteria_4 & v_{4j1} & v_{4j2} & v_{4j3} \\ c_{41} & 0 & 1 & 1 \\ c_{42} & -1 & 1 & 1 \\ c_{43} & 1 & 1 & 1 \\ c_{44} & 1 & 0 & 0 \\ c_{45} & 0 & 1 & 1 \\ c_{46} & 1 & 0 & 0 \\ c_{47} & -1 & 1 & 1 \\ c_{48} & 0 & 0 & 1 \end{array}\right]$$

Problem Profile =

$$\begin{bmatrix} Criteria_5 & v_{5j1} & v_{5j2} & v_{5j3} & v_{5j4} \\ c_{51} & -1 & 1 & 1 & \\ c_{52} & 1 & -1 & 1 & \\ c_{53} & 0 & 0 & 1 & 1 \\ c_{54} & 1 & 1 & 0 & \\ c_{55} & 1 & 1 & 1 & \\ c_{56} & 1 & 1 & 1 & \\ c_{57} & 1 & 1 & 1 & \\ c_{58} & 1 & 1 & 0 & \\ c_{59} & 1 & 1 & 1 & \\ c_{510} & 0 & 1 & 1 & \end{bmatrix}$$

Defect Profile =

$$\begin{bmatrix} Criteria_6 & v_{6j1} & v_{6j2} & v_{6j3} \\ c_{61} & 0 & 1 & 1 \\ c_{62} & 1 & 1 & 1 \\ c_{63} & -1 & 1 & 1 \\ c_{64} & -1 & 1 & 0 \\ c_{65} & -1 & 1 & 1 \end{bmatrix}$$

Market/User Profile =

$$\begin{bmatrix} Criteria_7 & v_{7j1} & v_{7j2} & v_{7j3} \\ c_{71} & 1 & 1 & 0 \\ c_{72} & 1 & 1 & 0 \\ c_{73} & 1 & 1 & 0 \\ c_{74} & -1 & 1 & 1 \\ c_{75} & 1 & 1 & 0 \\ c_{76} & 1 & 0 & -1 \\ c_{77} & 1 & -1 & 1 \\ c_{78} & -1 & 1 & 1 \\ c_{79} & 1 & 1 & 0 \\ c_{710} & 1 & 1 & -1 \\ c_{711} & 1 & 1 & 1 \end{bmatrix}$$

Management Profile =

$$\begin{bmatrix} Criteria_8 & v_{8j1} & v_{8j2} & v_{8j3} \\ c_{81} & 1 & 1 & 1 \\ c_{82} & 1 & 1 & 1 \\ c_{83} & 0 & 0 & 1 \\ c_{84} & 1 & 1 & 1 \\ c_{85} & 1 & 1 & 1 \\ c_{86} & 1 & 1 & 1 \end{bmatrix}$$

A.3 Scrum

Cost Profile =

$$\begin{bmatrix} Criteria_1 & v_{1j1} & v_{1j2} & v_{1j3} \\ c_{11} & 0 & 1 & 1 \\ c_{12} & 1 & 1 & 1 \\ c_{13} & 1 & 1 & \\ c_{14} & 1 & 1 & 0 \\ c_{15} & 1 & 1 & 1 \\ c_{16} & -1 & 1 & -1 \\ c_{17} & -1 & -1 & 1 \\ c_{18} & 1 & 0 & 0 \\ c_{19} & 1 & 1 & 0 \end{bmatrix}$$

Time Profile =

$$\begin{bmatrix} Criteria_2 & v_{2j1} & v_{2j2} & v_{2j3} & v_{2j4} \\ c_{21} & 1 & 1 & 1 & \\ c_{22} & 1 & 1 & -1 & \\ c_{23} & 0 & 1 & 1 & \\ c_{24} & 0 & 0 & 1 & \\ c_{25} & 0 & 1 & 1 & \\ c_{26} & 1 & 1 & 1 & \\ c_{27} & 1 & 1 & 1 & \\ c_{28} & 1 & 1 & 0 & \\ c_{29} & 1 & 1 & 1 & 1 \\ c_{210} & -1 & -1 & 1 & \end{bmatrix}$$

Team Profile =

$$\begin{bmatrix} Criteria_3 & v_{3j1} & v_{3j2} & v_{3j3} \\ c_{31} & 1 & 1 & 1 \\ c_{32} & 1 & 1 & 1 \\ c_{33} & -1 & -1 & 1 \\ c_{34} & 1 & 0 & 0 \\ c_{35} & 0 & 1 & 0 \end{bmatrix}$$

Development Profile =

$$\begin{bmatrix} Criteria_4 & v_{4j1} & v_{4j2} & v_{4j3} \\ c_{41} & 1 & 1 & 1 \\ c_{42} & 1 & 1 & 1 \\ c_{43} & 1 & 1 & 1 \\ c_{44} & -1 & 1 & 1 \\ c_{45} & 1 & 1 & 1 \\ c_{46} & 0 & 0 & 1 \\ c_{47} & 0 & 0 & 0 \\ c_{48} & -1 & -1 & 1 \end{bmatrix}$$

Problem Profile =

$$\begin{bmatrix} Criteria_5 & v_{5j1} & v_{5j2} & v_{5j3} & v_{5j4} \\ c_{51} & 1 & 1 & 0 & \\ c_{52} & 1 & 1 & 0 & \\ c_{53} & 1 & 1 & 1 & 1 \\ c_{54} & 1 & 1 & 0 & \\ c_{55} & 1 & 1 & 1 & \\ c_{56} & 1 & 1 & 1 & \\ c_{57} & 1 & 1 & 1 & \\ c_{58} & 0 & 1 & 1 & \\ c_{59} & 1 & 1 & 0 & \\ c_{510} & 1 & 1 & 0 & \end{bmatrix}$$

Defect Profile =

$$\begin{bmatrix} Criteria_6 & v_{6j1} & v_{6j2} & v_{6j3} \\ c_{61} & 1 & 0 & 0 \\ c_{62} & 1 & 1 & 0 \\ c_{63} & 0 & 0 & 0 \\ c_{64} & -1 & 1 & 1 \\ c_{65} & -1 & 1 & 0 \end{bmatrix}$$

Market/User Profile =

$$\left[\begin{array}{cccc} Criteria_7 & v_{7j1} & v_{7j2} & v_{7j3} \\ c_{71} & 0 & 1 & 1 \\ c_{72} & 1 & 1 & 0 \\ c_{73} & 1 & 1 & 0 \\ c_{74} & 0 & 1 & 0 \\ c_{75} & 1 & 1 & 0 \\ c_{76} & 1 & 1 & 1 \\ c_{77} & 1 & 1 & 0 \\ c_{78} & -1 & 1 & 1 \\ c_{79} & 1 & 1 & -1 \\ c_{710} & 0 & 0 & 1 \\ c_{711} & 0 & 1 & 1 \end{array}\right]$$

Management Profile =

$$\left[\begin{array}{cccc} Criteria_8 & v_{8j1} & v_{8j2} & v_{8j3} \\ c_{81} & 1 & 1 & -1 \\ c_{82} & 1 & 1 & 0 \\ c_{83} & 0 & 1 & -1 \\ c_{84} & 1 & 1 & 0 \\ c_{85} & 1 & 1 & 0 \\ c_{86} & 1 & 1 & 1 \end{array}\right]$$

A.4 Extreme Programming

Cost Profile =

$$\left[\begin{array}{cccc} Criteria_1 & v_{1j1} & v_{1j2} & v_{1j3} \\ c_{11} & 0 & 1 & 1 \\ c_{12} & 1 & 1 & 0 \\ c_{13} & 1 & 1 & \\ c_{14} & 1 & 0 & 0 \\ c_{15} & 1 & 1 & 1 \\ c_{16} & -1 & 1 & -1 \\ c_{17} & 1 & 1 & 1 \\ c_{18} & 1 & -1 & -1 \\ c_{19} & 1 & 0 & -1 \end{array}\right]$$

Time Profile =

$$\left[\begin{array}{ccccc} Criteria_2 & v_{2j1} & v_{2j2} & v_{2j3} & v_{2j4} \\ c_{21} & 1 & 1 & 0 & \\ c_{22} & 1 & 0 & 0 & \\ c_{23} & -1 & 1 & 1 & \\ c_{24} & 0 & 0 & 1 & \\ c_{25} & 0 & 1 & 1 & \\ c_{26} & 1 & 1 & 1 & \\ c_{27} & 1 & 1 & 0 & \\ c_{28} & 1 & 1 & 0 & \\ c_{29} & 1 & 1 & 0 & -1 \\ c_{210} & 1 & 1 & 1 & \end{array}\right]$$

Team Profile =

$$\left[\begin{array}{cccc} Criteria_3 & v_{3j1} & v_{3j2} & v_{3j3} \\ c_{31} & 1 & -1 & -1 \\ c_{32} & 1 & 1 & -1 \\ c_{33} & -1 & -1 & 1 \\ c_{34} & 0 & 0 & 1 \\ c_{35} & -1 & 1 & -1 \end{array}\right]$$

Development Profile =

$$\left[\begin{array}{cccc} Criteria_4 & v_{4j1} & v_{4j2} & v_{4j3} \\ c_{41} & 1 & 1 & 1 \\ c_{42} & 1 & 1 & 1 \\ c_{43} & 1 & 1 & 1 \\ c_{44} & -1 & 1 & 1 \\ c_{45} & 1 & 1 & -1 \\ c_{46} & 0 & 0 & 1 \\ c_{47} & -1 & 0 & 1 \\ c_{48} & 1 & 1 & 1 \end{array}\right]$$

Problem Profile =

Defect Profile =

$$\begin{bmatrix} Criteria_5 & v_{5j1} & v_{5j2} & v_{5j3} & v_{5j4} \\ c_{51} & 1 & 0 & -1 & \\ c_{52} & 1 & 1 & 0 & \\ c_{53} & 1 & 1 & -1 & -1 \\ c_{54} & 1 & 1 & 0 & \\ c_{55} & 1 & 1 & 0 & \\ c_{56} & 1 & 1 & 1 & \\ c_{57} & 1 & 1 & 1 & \\ c_{58} & 0 & 1 & 1 & \\ c_{59} & 1 & 1 & 0 & \\ c_{510} & 1 & 1 & 0 & \end{bmatrix}$$

$$\begin{bmatrix} Criteria_6 & v_{6j1} & v_{6j2} & v_{6j3} \\ c_{61} & 1 & 0 & 0 \\ c_{62} & 1 & 1 & 0 \\ c_{63} & -1 & 0 & 1 \\ c_{64} & -1 & 0 & 1 \\ c_{65} & -1 & 1 & -1 \end{bmatrix}$$

Market/User Profile =

$$\begin{bmatrix} Criteria_7 & v_{7j1} & v_{7j2} & v_{7j3} \\ c_{71} & 0 & 1 & 1 \\ c_{72} & -1 & 1 & 0 \\ c_{73} & 1 & 1 & 0 \\ c_{74} & 1 & 0 & 0 \\ c_{75} & -1 & 1 & 0 \\ c_{76} & 1 & 1 & 1 \\ c_{77} & 1 & 1 & 0 \\ c_{78} & -1 & 1 & 1 \\ c_{79} & 1 & 1 & -1 \\ c_{710} & 0 & 0 & 1 \\ c_{711} & -1 & 1 & 1 \end{bmatrix}$$

Management Profile =

$$\begin{bmatrix} Criteria_8 & v_{8j1} & v_{8j2} & v_{8j3} \\ c_{81} & 1 & 1 & 0 \\ c_{82} & 1 & 1 & 0 \\ c_{83} & 1 & -1 & -1 \\ c_{84} & 1 & 1 & 0 \\ c_{85} & 1 & 1 & 0 \\ c_{86} & 1 & 1 & 1 \end{bmatrix}$$

A.5 Feature Driven Development

Cost Profile =

$$\begin{bmatrix} Criteria_1 & v_{1j1} & v_{1j2} & v_{1j3} \\ c_{11} & 1 & 1 & -1 \\ c_{12} & 0 & 1 & 1 \\ c_{13} & 1 & 1 & \\ c_{14} & 0 & 0 & 0 \\ c_{15} & 1 & 1 & 1 \\ c_{16} & -1 & 1 & 1 \\ c_{17} & -1 & 1 & 0 \\ c_{18} & 1 & 1 & 1 \\ c_{19} & 0 & 1 & 1 \end{bmatrix}$$

Time Profile =

$$\begin{bmatrix} Criteria_2 & v_{2j1} & v_{2j2} & v_{2j3} & v_{2j4} \\ c_{21} & 0 & 1 & 1 & \\ c_{22} & 0 & 0 & 0 & \\ c_{23} & 0 & 1 & 1 & \\ c_{24} & 0 & 1 & 0 & \\ c_{25} & 1 & 1 & -1 & \\ c_{26} & 1 & 1 & 1 & \\ c_{27} & -1 & 0 & 1 & \\ c_{28} & 1 & 0 & 1 & \\ c_{29} & 1 & 1 & 1 & 1 \\ c_{210} & -1 & 1 & 0 & \end{bmatrix}$$

Team Profile =

$$\begin{bmatrix} Criteria_3 & v_{3j1} & v_{3j2} & v_{3j3} \\ c_{31} & 1 & 1 & 1 \\ c_{32} & 0 & 0 & 0 \\ c_{33} & 0 & 0 & 1 \\ c_{34} & 1 & 1 & -1 \\ c_{35} & -1 & 1 & 0 \end{bmatrix}$$

Development Profile =

$$\begin{bmatrix} Criteria_4 & v_{4j1} & v_{4j2} & v_{4j3} \\ c_{41} & -1 & 0 & 1 \\ c_{42} & -1 & 0 & 1 \\ c_{43} & 0 & 1 & 1 \\ c_{44} & -1 & 1 & 0 \\ c_{45} & 0 & 0 & 0 \\ c_{46} & 1 & 1 & 1 \\ c_{47} & 0 & 1 & 0 \\ c_{48} & -1 & 1 & 0 \end{bmatrix}$$

Problem Profile =

$$\begin{bmatrix} Criteria_5 & v_{5j1} & v_{5j2} & v_{5j3} & v_{5j4} \\ c_{51} & 0 & 1 & 1 & \\ c_{52} & 1 & 0 & 1 & \\ c_{53} & 1 & 1 & 1 & 1 \\ c_{54} & 1 & 1 & 0 & \\ c_{55} & 1 & 1 & 0 & \\ c_{56} & 1 & 1 & 1 & \\ c_{57} & 1 & 1 & 1 & \\ c_{58} & 1 & 1 & -1 & \\ c_{59} & 1 & 1 & 0 & \\ c_{510} & 1 & 1 & 1 & \end{bmatrix}$$

Defect Profile =

$$\begin{bmatrix} Criteria_6 & v_{6j1} & v_{6j2} & v_{6j3} \\ c_{61} & 1 & 1 & 1 \\ c_{62} & 1 & 1 & 0 \\ c_{63} & 0 & 1 & 0 \\ c_{64} & -1 & 1 & 1 \\ c_{65} & 0 & 1 & -1 \end{bmatrix}$$

Market/User Profile =

$$\begin{bmatrix} Criteria_7 & v_{7j1} & v_{7j2} & v_{7j3} \\ c_{71} & 1 & 1 & -1 \\ c_{72} & 0 & 0 & 1 \\ c_{73} & -1 & 0 & 1 \\ c_{74} & 0 & 0 & 0 \\ c_{75} & -1 & 1 & 1 \\ c_{76} & 1 & 1 & 1 \\ c_{77} & 1 & 1 & 1 \\ c_{78} & -1 & 1 & 1 \\ c_{79} & 1 & 1 & -1 \\ c_{710} & 0 & 1 & 0 \\ c_{711} & 1 & 1 & 1 \end{bmatrix}$$

Management Profile =

$$\begin{bmatrix} Criteria_8 & v_{8j1} & v_{8j2} & v_{8j3} \\ c_{81} & 1 & 1 & 1 \\ c_{82} & 1 & 1 & 1 \\ c_{83} & 0 & 1 & 1 \\ c_{84} & 0 & 0 & 0 \\ c_{85} & 1 & 1 & 1 \\ c_{86} & 1 & 1 & 0 \end{bmatrix}$$

Appendix B. Interview Protocol

[Introduction]
My name is Shibani Basava. I'm a Masters Student at CU Boulder and I'm currently working on my thesis. I would like to interview you as the manager of [Team Name] at [Company Name] to learn more about the team and development practices in a professional organization for my Masters Thesis.

[Overview]
I am interested in learning your problem domain, your team characteristics, development practices, tools you use and other things that influence your process. For my thesis, I am trying to find out how to determine which development life cycle and/or tools best match a given software development team or project. Your input will be really valuable in understanding current practices in the software industry.

This interview will take approximately 45 – 60 minutes of your time. I would like you to sign a consent form before participating in the interview. Any information you provide will be completely anonymized and no one other than my advisor and me will have information about your participation. I will be audio taping the interview in order to accurately record the information. If you are not comfortable with that, please let me know and I will not record this session. These audio records will be erased once the data has been anonymized and recorded.

Do you have any questions before we proceed?

[Consent form]
Here is the consent form giving me permission to interview you. Please read through it thoroughly. There are two copies of the consent form; one is for you to keep for your records. Please initial each page of the consent form and sign the last page.

[Interview questions – These are open ended questions and will lead to discussions]
If you are ready, let's proceed…

- Starting off… can you tell me more about what you do? Your problem domain? What issues are you trying to address? Do you directly interact with your customer?

- How big is your team? What are the different roles on your team, e.g. Testers, Developers etc.? What kind of interactions do they have and how do they have it?

- What development life cycles do you employ? Why do you use this lifecycle for your team? Does your team size or structure or project needs influence the lifecycle?

- How long does your project last? Or how often do you make deliveries to your customers? How do you manage these deliveries? How do you track the timeline such that it is finished within the scheduled time? What do you do if the schedule is not met?

- Can you describe the workflow for each release/iteration you define?

- In this workflow, do you use tools to enhance the process? Where? What are they? What do you use them for? Why do you need them? How often are they used? Who uses them? Have you ever run into problems using them?

- What kinds of tools do you use during your implementation phase to assist QA and reliability of each delivery and testing?

- What kind of development - testing ratio do you have? When does your testing phase fall into your lifecycle?

- Do you communicate with other teams? What communications do you do with the other team? How do you communicate?

- How do you gather your requirements? How do you manage them?

- Do you think this is the best development process for your team/project? Why? If not, what process would you like to use?

- Do you use object-oriented design? UML?

- What languages do you use for development? Do you use an IDE for assisting in the development? How effective do you think it is?

- How do you manage traceability between requirements to design and design to test? Do you use tools to manage this? If yes, what tools do you use? If no, why?

- How do you perform testing – Manual or Automated? If it's automated, what tools do you use? Are they off-the-shelf or custom made? If custom, why was there a need to create a custom tool? – was there no equivalent available on the market? For manual testing, how big is the QA Team? Is this size good compared to your development team size?

- How do QA and development communicate with each other? How are bugs tracked? How is the fix communicated back to the QA? Do you think this is the best method? Why?

- Are there any gaps in the development process that you think you were not able to address because of a ready made tool, or because of the project/team characteristics? If so, how do you address them?

- What characteristics do you need from your development process – Traceability, Reliability, Verification, and Usability? How do you achieve these objectives?

- Are there any external entities that influence your process – e.g. FDA? How do they influence the project? How is this influence handled? Do they affect the design process heavily? How about the testing?

- How formal or informal do you think your development process is? Why? Can it be improved? How so?

- How are inter team communications managed? Do you perform regular meetings? Team building?

- What other tools do you use to assist your development process? Why do you use them?

[De-briefing]
Thank you for your time today. I would like to reassure you once again that any information you have provided will be completely anonymized in the final report. [If the audio has been recorded] The audio will be erased once notes have been recorded. The results of this interview will be documented in my Masters Thesis.

If I have more questions in the future regarding the information, is it okay to contact you? Thank you.

If you have any questions/concerns regarding this interview please feel free to contact me.

Once again, thank you for your time today.

Appendix C. Interview Transcriptions

C.1 Interview A

Starting off… can you tell me about your problem domain?

We are in the Research and Product Development department of a Medical device company and we develop a product which is an interior architecture software application with various different technologies involved – database backend, different architecture – there's the client server, web services, web applications and service oriented architecture as well.

What issues are you trying to address with this product?

The reason our product exists is to help our customers manage collections of blood components. We are primarily a medical device company, so we have a flagship device that does automated collection of blood components. The software product that we develop actually takes that to the next level and helps automate the entire collection process from the time the donor walks in, throughout the donation itself and looking at it overtime historically to determine eligibility etc.

Do you as manager directly interact with your customer?

I do interact directly with customers, that's not my primary responsibility and I have a cross-functional team that I'm a member of, and on that cross-functional team are directly customer facing individuals such as sales, marketing and our implementation and support.

You mentioned the cross-functional team. Can you explain more about what the responsibility of the cross functional team is?

Sure. As the title dictates, cross-functional team is a team that's responsible for a project and it represents the different departments or functions of our organization. So, it's that multi-function or multi-facet team and the responsibility of that is to use the power of various points of view to do the right thing for our product and for our customer.

How big is your team?

Let's see – my development team is made up of four developers, 2 QA members, an additional QA resource that's focused on test automation and then myself, the software manager.

Do you think that this size is good for this project? Why?

I wish it were bigger. There is a lot of demand and lot of ability for us to develop and continually evolve our product. There are a lot features we can certainly grow and bring a lot of return on our investments. However, we’re not the only product, not the only priority in the organization. We have a lot of innovative ideas. We have well over 40 different active projects all needing to be prioritized within R&D so it’s an understanding. We all would like to have bigger teams and more resources.

What development lifecycle do you use?

Actually here at this company we have what we refer to as a Spiral model of development and that might be slightly different from the traditional software spiral modeling definition but very similar. But in effect what its saying is that we’re not developing a single product and putting it into a maintenance mode; we’re developing the product and our work then continuous on with the next spiral, which is our next better version etc. feeding back what we learned from the previous spiral.

How is it different from the traditional spiral model?

The traditional spiral model, I think fits more on the internal product or internal application. We have fairly tight spiral numbered in a few weeks. In the ideal case you have a customer, the end user as part of the cross functional team. In this application, in this team we’re developing products so it’s a larger scope. We might actually have some of the internal spirals within a release; when I was referring to the spiral model earlier it’s more of the macro view of that.

So how long does one spiral last?

Major releases, and a big part of this is driven by our ability for our customers to take on new challenges and new versions within a highly regulated environment, so really, in terms of spirals we’re looking at one year to 18 months. I think I also mentioned that my development team is also responsible for maintenance releases of fielded tasks. So we’re not always focused on just the new version, we also need to be providing incremental updates and fixes and features to our fielded versions as well.

How do you track the timeline such that it is finished within the scheduled time?

The big challenge there is that being that it is a long term, it’s bound to change. So from the time you say 12 to 18 months from now, this is the product that we want to have released in the market, that’s a lot of time. And so half way through, new requirements might have evolved. The market might have changed etc. So I think really what we are looking at is recognizing that we can’t release faster than that because the market can’t accept it unless we can reduce the impact to our customer when we have a new feature. So that’s one of the things that we have taken on from the technology standpoint. And there are a couple different mechanisms we use to track the time. We have a thing that we call a product map, and more or less what you might guess it is. It’s owned by marketing, but certainly developed and kept in synchronization with R&D. Basically, what that product map is meant to do is be a mechanism to make sure that marketing is in sync with product development. This product map works at the management level to define at high level what our next releases are for the next 3-5 years. So right now our product map goes out to 2012-2013.

Secondarily, as we get into the different phases, beyond the exploratory prototype, we leverage more detailed development schedule which is picked out down to a week or less of the incremental time and in the ideal situation that helps us manage, when we think we're [going to] get done. The variable is always what happened in the field is going to continue to change the requirements significantly or what do we need to react to that's going to change the priority.

What do you do if the schedule is not met?

That's kind of the classic management, what are your… what tools or knobs you have to turn and I don't think there's particularly unique about our situation, fundamentally there's a handful of things you can do if you are falling behind schedule. I guess one of the things that I should mention is we do have a very aggressive and optimistic take on scheduling here, one of the things that's kind of a fundamental aspect of R&D, driven straight down from our vice president is what he terms 20% schedule. He likes to work towards a 20% schedule. That's unique. I've never encountered that in any other organization I've been involved with. And basically his perspective on that is let's create that schedule as if everything is perfectly right, and ideal situation you're [going to] make that date. In reality very rarely are you going to make that date. His thought beyond that is so you miss the 20% schedule, but you end up because of that aggressive aspect of always looking for the best way to solve the problem and asking those questions earlier, you discover the problems much earlier and end up beating what would have been the 20% schedule.

Do you communicate with other teams? What communications do you do with the other team?

Absolutely; there are a two different ways to slice that. We share the same market space with our main device and therefore our product actually has the electronic communication with that device to an interface. So obviously we need to stay in sync with that. There's couple different ways of doing that - during the development phase, we have an interface document that manages that and members of my development staff work directly with members of the device development staff to stay in sync and understand that low level detail. We rely on the management team – myself and my colleagues at that level to understand what's changing in the device's future versions, what's potentially going to impact the next of our product. So those are the 2 ways of staying in sync with the device that we communicate with. The other aspect is for similarly architected applications. We have another product or two that may not be, is logically separate from our product, and doesn't communicate with it. However, we want to share ideas, share architecture, share solutions for some of the problems. And again that happens at both levels, both at lower level detail between the developers but also at the management. I think the other thing that is advantageous about our company with that is we don't fix our teams too much, so we have members of my team removed to other teams and vice-versa so we have cross pollination which helps also understanding the technology and understanding what problems been solved.

How do you communicate between teams?

Being that we are product focused measured on our ability to get the product out, deliver a product that meets our customer needs, that's our primary focus. The communication between 2 different development teams is not something that we have formalized, the communication process. So I think there are 2 mechanisms – one is the informal face to face communication and a way that we help facilitate that is to co-locate these similar technologies in the same space. So if we have 3-4 teams that are working on the same technology we can benefit from each other, we're all on the same floor across the same area. The other mechanism is we use SharePoint as our repository for design and deliverable documentation and that SharePoint is the force that can be used to across teams.

It's used on a daily basis; it depends on what phase of development we're in and whose using it on a daily basis. However some of us definitely use it on a daily basis, something that is essential for intra team communications and maybe used on a weekly basis for inter team communication.

How do you gather your requirements? How do you manage them?

Since we are a medical device company that is regulated a lot of the things that we do by its nature we have a quality system that documents it. So we have SOPs, Standard Operating Procedures that provide if not detailed instructions on how to dos and policies. The SOP that we use, we nick name is the 007 and it's our SOP for developing a new product or significantly modified product. So it goes through several stages, first is the design plan, second is the design inputs. The design inputs are fundamentally our requirements gathering and analysis stage. It depends on the product and it depends on the phase and maturity of the product. In the case of our software system, this was originally almost unprecedented for our company; so nearly a decade ago when the mission was launched it was worked initially on with experts that understood the medical device and understood the gaps at a high level that our customers were facing, and a lot of times our customers didn't know the gaps, didn't know that kind of a thing. So we developed a fully functional beta version and took that out to the field for extended beta trial in 2 different locations in the US and also in our other European markets as well. Since then our product has matured and gotten more sophisticated as well, the requirements gathering has come from multiple sources. All of our customer facing individuals have an opportunity to bring this to the cross functional team. And that's probably another note for the tool that we use, and essentially for all software development is our Issue Tracking system. A lot of times the issue tracking system is only associated with defects. The way we use the system is bug tracking, defects as well as enhancements. So I guess what I would say is 3 main ways the requirements come in – 1) understanding changes to our device we are married to; 2) d etc; and then 3) a continual review of the issues that are in our Issue Tracking system, both defects and enhancements. And that's another aspect, a sub team of our cross functional team which we call the IT Review Board has responsibilities for assessing the issues – deferring them or assigning them to different releases.

What development life cycles do you employ within each Spiral? How do you employ it?

Fundamentally we use something that's very similar to traditional waterfall. We have another SOP for software development that outlines that. At a high level we look at analysis and planning, requirements, design, unit test, integration, verification and validation. So that would be our typical cycle. One thing that I would also put into that is depending on the complexity and risk uncovered during the analysis phase, we would take on prototyping or another term we use is spike for understanding what we are capable of and how we are [going to] solve problems that aren't obvious.

Did you need to adjust the life cycle from the traditional idea to suit your project needs? If yes, how so? If no, why not?

There are variations to the waterfall process, and coming from an Aerospace background, when I say waterfall, I think of a very specific regulation for what a waterfall is, and I'm not absolutely sure that we meet all of those steps. For instance another variation on the waterfall that I'm familiar with has design put through preliminary design, critical design, all of which have milestones, or interim milestones, where typically our milestones would be one requirement milestones.

I think it's a fallacy in practice to freeze stages of the waterfall process, I'd never seen a waterfall that actually freezes ever. Even with the preliminary design, I think no matter how good your design changes, there's always enough noise in the system and things that you learn, whether it's a new requirement that [kind of] creeps in because nothing happens instantaneously. Or a new technology that is better suited to the needs or the developer learning and improving overtime, all of those things cause some ripple. The key to success is to maintain and minimize the impacts, so that you don't get locked up never being able to complete a stage and always just falling back to the requirements and actually getting out of your waterfall.

Yes, I had to make changes to the process for our project teams. I think I've mentioned in our development cycle, one of the things that have a lot more emphasis in our market space than others is the regulatory aspect and a safety aspect. So we have additional tollgates, if you will, to go through in terms of risk analysis. We use a full analysis process called FMECA which stands for Failure Mode Effects Cause Analysis. And then we also have regulatory requirements for being able to release into certain markets. Such as, I think the most typical one is 510 K approval for the FDA to really say that the device that has very similar to already certified. So those are the steps we had to add to the traditional waterfall.

Do you think this is the best development process for your team/project? Why? If not, what process would you like to use?

Well I think it works, we got a system that works, we develop a product that we have pride in and meet our customer needs. So I think its an effective development process. Personally just because of my own style is, I'm not going to say this is the best, because I don't know what I don't know and I always think there's room for more improvement.

I think depending on the problem and the issues at hand, we've actually done variations within our own process as well. There are certain areas that I have hinted at earlier if you got more risks, you put more emphasis on prototyping. And when I've worked with a slightly smaller team with a smaller scale product, we had a very good success at identifying several small iterations of the traditional spiral to help us do those checks required for the final release. So I think there are aspects of different processes that we can bring to bear. Another thing that brings to mind is that we've done is we've pulled some of the agile methodologies in the past doing peer programming, which was very interesting but we had a release that we couldn't get wrong and we didn't have, because of the nature didn't play very well with black box testing, so we needed to have more emphasis on getting code right the first time. So having four eyes on it instead of two was effective in the short run. Puts a lot of stress on the developers though, mostly its uncomfortable for most people and hardly justified from the management perspective because the perspective initially is that I don't want two people doing one person's job.

Do you use object-oriented design? UML?

Yes.

What tools use for UML design?

Over the history of the project, we've used primarily three different tools for UML modeling, diagrams etc. We started with Rational Rose and I think the problem with that was the promise of keeping basically the code generation and being able to go full cycle forwards and backwards, reverse code generation. Sounded a great theory but didn't work very well in the original. Second and most prevalent is Visio. Visio has decent tools for reverse engineering existing code. However, it's used primarily prior to code step as we are identifying the key classes. And this is where I'll enter a little bit of my own opinion in terms of UML modeling and tools. In my opinion the strength of doing modeling is the abstraction of the key essence of the problem you are trying to solve. And if you try to use UML modeling to actually generate code, you lose the trees from the forest. Because now you have to put in all this detail into your model and it's no longer abstract, it's concrete. That's a challenge. On the other hand if you don't put those details in, you quickly find your design artifacts go out of sync with your actual code base. The third now is recently with .NET Visual Studio 2005 has its own pseudo UML modeling, class diagram generation right within the development environment and heard great things at first with that. I think its good at that abstract level but when you finally start hitting the code construct mechanism, I think it might fall short again.

Can improvements be done to UML Modeling?

Again, I think the modeling is best suited for the abstract that talks about the key interaction, the key relation of our system and communicate those things. And I think UML class diagrams are a fantastic way of showing one angle of that, I think sequence diagrams are another view into the problem domain and can be very helpful. And then depending on the problem, state diagrams are different, and sometimes can be a much better design artifact than either of those things depending on the problem solved.

What languages do you use for development? Do you use an IDE for assisting in the development? How effective do you think it is?

We use primarily C++ and C#, our initial releases of the product were in Visual C++. C# followed on, pushed into a couple of modules, now slowly and surely having its impacts into the core aspects of the system as well. We use PL/SQL because we have an Oracle back end, we have certain bits of project and functionality that's captured in stored procedures and functions and objects. We have InstallShield which is a specialized tool for developing installers. It has its own scripting language. And a little bit on the edges we do some Java Scripting on our web applications. That's the high level. There are few other scripting languages that might be used for build tools and code generation, but fundamentally the application's developed in the languages I've already mentioned.

Visual Studio and PL/SQL developer, I think Visual Studio is probably best of breed for our development environment. Having come from an embedded world when I first got exposed to even the previous releases of Visual Studio, I felt like I was cheating from the set of tools at hand. Like any tool, though, in order to be effective with it I think our developers know how to use the tool.

How do facilitate this training?

That's probably an area for improvement in my own team. So right now we don't have a formal set of training with the full IDE. However it is a pre-requisite of when we bring team members on to have the experience with that or have experience it feels very relevant such that it can be learned on the job. The other thing that I expect from our team is a continual communication between members so that we can develop those ad-hoc best practices to make sure that we're taking advantage of the various assets of the IDE as well as the language. That's part of the expectations on the team as well.

How do you manage traceability between requirements to design and design to test? Do you use tools to manage this? If yes, what tools do you use? If no, why?

Yes, we … again being a regulated environment one of the important things we have in our design artifacts is traceability. We use a tool called Rational Requisite Pro for the majority of our traceability. Because we're driven from the regulated environment we've identified what our key traces are and at a high level we take design inputs to system specifications, system specifications to SRS, SRS to test, test to verification. Those are the key traces that we go through and use the requisite pro tool for. In addition something that we've always had an expectation on but not last year or so had a process in place to actually really trace the requirements into the design itself. And we looked at how best to leverage the tools for that and at this time it's basically determined that we expect a manual trace from requirements to design and that's part of the deliverable when we do a design review.

How do you perform testing – Manual or Automated? If it's automated, what tools do you use?

The automated testing started out as a "hey let's see if we can see if there's any value on that." An internal person who had primary responsibility elsewhere and as secondary or tertiary responsibility took on developing a test automated system. And during that phase, the tool that was selected was Rational Robot. However, Rational definitely has some drawbacks, and when you get into 3rd party tools for UI control, Rational is definitely behind the times. So relatively recently this calendar year, we've taken on additional tools and that we're looking at using in addition to Rational Robot, the QTP, an HP product, which used to Mercury Quality Test Pro, but now is HP Quality Test Pro. One thing that we also did is brought in some consultants that had extra experience in test automation and have asked them to help us provide a best practices and process. One of the take away from that endeavor is that theirs isn't one tool that's perfect, not this tool is better than this tool, this tool does something better and other tool does other things better for a lot of multi technology systems like our product, we may end up with a lot of test automation tools in combination of. But more important thing is actually having a good framework for the way that the test automation engine is developed. I think that's more important than the test automation tool itself. We've a high level framework that we're working to and I think its lending itself to repeatability and maintainability of the test automation.

Are they off-the-shelf or custom made? If custom, why was there a need to create a custom tool? – was there no equivalent available on the market? For manual testing, how big is the QA Team? Is this size good compared to your development team size?

Primarily we're leveraging off-the-shelf tools, however we have some in house mechanisms to [kind of] tie the whole process together, we have a tool that ties the automated build, deployed automatically and then tied into a subset of our automated tests, we call it the V-BITE system. We leverage several tools so we have a little bit of tying existing tools together, i.e. development environment to the build, scripting tools that help tie things together. The integration tools are definitely unique to our products, so there definitely wouldn't be an off the shelf tool for that. And then tying it to our Rational Robot, so the majority of the work is in integrating these off-the-shelf tools.

How big is your manual testing team?

We have depending on the day 2 or 3 QA members. Their focus and expertise is primarily on Black Box testing with a little bit of White Box testing, but their expertise is certainly on Black Box. In my experience we got about the right size QA for our current development size, I think 60:40 is about right.

How do QA and development communicate with each other?

We are an integrated team. One of the things that I drive towards is a balance. On the one hand we want to have a QA function that is separate from development, so they actually have a reporting structure that is matrix'd, they are functionally reported into a QA management team separate, however, from the project standpoint, I've drove hard to integrate the development and QA team to just one team. The development and QA team will have regularly scheduled weekly meeting for a formal gathering where we can talk about project stuff. But I absolutely depend on daily interaction between our developers face to face. Again co-location is key, having teams that are working on the same project have to close to each other to have that kind of interaction and then finally the other way the team can interact is through our Issue tracker system.

How are bugs tracked?

The Issue Tracker system is primarily internal system; there are other systems in place that are used for managing customer interactions and also customer complaints. Our issue tracker system is not the key system that's leveraged for that. From the external perspective enhancements will effectively filter into the issue tracker system and the IT system is primarily communicating into our product development team.

How is the fix communicated back to the QA?

That is through our process and through our issue tracking system. So the developer is responsible for updating the issue, setting it to a new status, which becomes 'Resolved, Aviating Confirmation' and then assigning it back to the appropriate waiting QA team member. In addition our QA lead for the team is continually watching over the open issue trackers and watching when things does get solved, so if it doesn't get assigned to a QA person, then he can do that.

Do you think this is the best method? Why?

See my previous comment, although I do think this is effective.

Are there any gaps in the development process that you think you were not able to address because of a ready made tool, or because of the project/team characteristics? If so, how do you address them?

I think there are 2 things that come to mind. One is about the developer's continually be able to know how to best use their tool. So there's potentially a gap on the way that we are using our tools, something that we haven't put into formal training consistently. Certainly we have done training on aspects of our tools, one that comes to mind is Infragistics, and earlier the team has come across. And we control some pretty sophisticated capabilities, so we have a majority of the team going to get trained on. I think the other thing that I can see an area for improvement is with our team, I've done this with a team in the past, with good results, but I haven't done with this team is to take it to a sports analogy in order to be successful in winning at a sport, whether it be football or Tae Kwon Do, there's a concentration on fundamental. You've got to always go back to the fundamentals, you can't learn the fundamentals and just concentrate on the expert stuff. And one of the things that I think would be valuable to the team is to have some review and continual kind of back to the fundamental, yes I know this but let me make sure that I'm sharp on it. When I'm talking about fundamentals its really the object oriented stuff. The best practices for that, what is the way to break down problems, what makes software great? Asking those fundamental questions should be obvious but also should be refreshed all the time.

What characteristics do you need from your development process – Traceability, Reliability, Verification, and Usability? How do you achieve these objectives?

All of those actually. If you want to have a good software, it has to meet the customers needs. In order to meet the customers needs it has to be useful, so that's an important things, that's something that we rely on the cross functional aspect of our team to make sure that when we are going through our concepts and coming up with a User Interface that its not too developer-centric. Quality is our differentiator, that's what we are known for. And that plus being able to support and when the inevitable bug is impacted how we'll react and make it right by the customer. So those are the things that we focus on and to me traceability and regulatory aspects are mechanisms for achieving those other higher end goals.

Are there any external entities that influence your process – e.g. FDA? How do they influence the project?

In the US, it's certainly the FDA, but we also work with a British Standard Institute, BSI, that comes and performs audits on us on regular basis so that we can maintain our own ability to market our products in Europe. So those are the two big external bodies.

How is this influence handled? Do they affect the design process heavily? How about the testing?

At a high level, yes. From the get go as I mentioned our design process one of the drivers for that is that we meet and being able to communicate to the auditors what we do. When we identify the deliverables for our project or for our spiral best cases that we should be thinking about ‘Is this going to meet the need when the auditors come in’. At the same time, I don’t want to create work for work’s sake, although its something that we need to have for the auditing. If we’re doing it right its something we need to have anyway, lets make sure we have a high quality product.

How formal or informal do you think your development process is? Why? Can it be improved? How so?

That’s a relative question, I think we’re much more formal than the typical internal application development, IT [kind of] stuff. Probably by an order of magnitude. I say we’re on par with other federal government contractors. Interestingly we’ve seen in our personnel that we’ve hired at the management level, there’s a co-relation, we’ve several of our managers that come from AeroSpace background. And I think part of the reason is their ability to achieve innovation within a more regulated or formal environment.

How are inter team communications managed? Do you perform regular meetings? Team building?

My philosophy on the inter team thing is that it’s essential to our success. So I want to be able to have an informal work environment where people are happy and want to come to work and at the same time know why we’re here. Its customer focused and regulatory. In order to do I certainly encourage communications between the team members on the regular basis, try to set up opportunity so that we’re not separate, so that from a particular aspect of the system, we don’t have just one person know it. So there needs to be technical communication between that and then we have the regular team meetings and finally with the majority of the team I have a weekly and bi-weekly one on one meetings.

C.2 Interview B

Starting off… can you tell me more about what you do? Your problem domain? What issues are you trying to address? Do you directly interact with your customer?

I am the software lead for a project which is a therapeutic aphaeresis platform, so we treat various blood diseases. Have a team of 8 people, including myself. We tend to work on multiple protocols at the same time which makes it a little bit interesting. Right now we’re working on Therapeutic Plasma Exchange, Red Blood Cell Exchange, and White Cell collection. So, we’re working on 3 different parts of the project.

Code reuse and code train becomes big issues for us. We obviously don't [want to] have to maintain substantially similar versions of the same thing in multiple places. That's just asking for a bug in one that we fix there but forget to fix in other places, so we spend fair amount of time trying to identify common pieces in all the protocols. We're looking at putting them in a common location and not have to maintain copies of it. It's a manual process right now. I have looked at using this tool that can compare text files and locate files where there are large areas of similarities and so I looked into using that and it looks a lot promising. Right now I just haven't had time to try it out more than look at its page and it out to be a useful thing.

How big is your team? What are the different roles on your team, e.g. Testers, Developers etc.? What kind of interactions do they have and how do they have it?

8 people. That's just the developers. The testers work under another person and there are probably about 12 members in the team. The teams work out pretty well, I think we're getting to the point where development is starting to outrun our testing capabilities. So we might need to expand the testing to take care of that. We're starting to be more limited by our testing than by our development.

Right now, our releases are being gated not by when we have the final software ready for release testing but how its [going to] for testing. It wasn't that case to begin with, the first release we had development and testing running pretty much in parallel and not at different schedules. We started branching and developing multiple protocols and started, I think what's going on is that we have identified common areas in software and we can do a lot of things in common that won't require re-development and re-protocols. Unfortunately they are still testing to see if those common things work with the new protocols. So the fact that we have a lot of common code speeds us up but it doesn't speed up testing as much. Can't skip some tests but they can skip as many tests as we can skip writing modules.

How long does your project last? Or how often do you make deliveries to your customers? How do you manage these deliveries? How do you track the timeline such that it is finished within the scheduled time? What do you do if the schedule is not met?

The entire project is scheduled out to release in 2010, I believe. So, its started well I guess 2003/2004 were the first time with marketing studies. Development started in 2005 in earnest, I mean there were earlier prototyping stuff for marketing purposes but the actual development started in 2005.

My little paper schedule. Basically we manage it by breaking the overall development into smaller pieces that we can deal with. I tried to schedule the development cycles such that we have deliverables in a 9 month time frame which is to be relatively right amount of development time. We can keep everybody on track and know that we're getting off track and know that we are behind schedule if we are. Its like you mentioned if we are doing a 5-year timeline its very hard to difficult to know, to determine where you are on the timeline. So we break it into more manageable pieces. Which works well since we have so many protocols in the machine, the protocols make a natural separation.

We certainly are trying [to give] them tested deliverables. We generally have three phases of release per protocol, and then there will be some maintenance versions after that. But we will release the first clinical per protocol and that will be the first to see human use. We release an updated one with what we learned with the prototype that goes more into routine use so it's a wider use under less our control. And then finally the production version. Therapeutic plasma exchange gone through all three of those and then we have a maintenance release for TP scheduled for February of next year. Some protocols are quite simple, we won't have all those deliverables, for example, bone marrow processing probably won't have a clinical release because its something we can test entirely in-house. We don't need to have that intensive testing in the clinic. So as long as you have a person attached to the person, you can process it.

Our R&D VP very much likes to see extremely aggressive schedules. His basic guideline is that your basic schedule should be written assuming that everything will go well and he expects that schedules will be missed 80-90% of the time. But when they are met you obviously seems to work on it pretty well. And it seems to work out pretty well. The important thing or a couple of important things about missing schedules is communicating to everybody, sometimes repeatedly that you're working with an aggressive timeline and not meeting a schedule isn't a failure. It means that we learned something along the way and that we have developed a better product for it. The second big part is communicating with all the other groups to make sure that other people that are depending on our deliverables know that things are [going to] be late. A big one that's coming up a lot right now is the translation group. We have to deliver stable string files to the translation group to have them translate it. But obviously if our deadline slips, then we can't get them string files to work with so we have to let them know. Testing is another group where its really important, Disposables, we have to have disposables to do the clinical trials.

How do you facilitate the communication?

We have a weekly meeting to discuss issues, so once we go through all the new issues for that week, in that meeting we have representatives from the technical writers, we have clinical people, and disposables in the clinical group and electrical as well as software and QA. So that tends to be a major meeting where everybody can be up to date on what the challenges are and what's happening and what things are being pushed back. In addition to that I publish one of these updated timelines about every three months, and I try to schedule it out for about a year and a half. So every three months I deliver to everyone on the timeline that the main things I show on my time line are software developers time, QA time and when we are expecting translation drops to the translation group. But the schedule also shows when we are expecting clinical trials to happen. So its [kind of] the state the software is as we currently know it.

How do you publish it?

I do it the old fashioned way. I email it to people. I usually, after I do that I have a meeting with my supervisor so the project manager for the system, and we do is make changes to the schedule if we thinks its necessary. For example, the project manager has all the detail information on all the clinical aspects of the system and whether we're going to get into clinical trials later than we expect. So usually there's some modification to the timeline at that point but when we're done I email it out to everybody that's interested. I have a meeting with my software developer group to go into detail.

How do you manage new requirements?

Well that is partly what we discuss at the weekly meeting, the way we get new requirements into the system is through the Issue Tracker, it's the same way that we get defect reports into the system. So at those meetings if there have been new requirements, we'll discuss them. The primary things to come out of that meeting are whether or not the requirement is brittle, sometimes we say no we're not [going to] do that. If we decide we are going to accept it we want to make changes to it, we might assign somebody to investigate it and see how hard its going to be to do and come back later. But alternately once we have accepted it and decided what we are [going to] do with it we assign it to one of the software releases. Then decide which one its going to. Things that control that are things that have a major impact on usability and they are given high priority. Obviously anything related to safety is given the highest priority and we do actually have one situation where we actually pause the development to address this new requirement found in the field. So sometime the issues just completely block the normal development and just require everyone to work on one aspect of the issue at a time.

What's the issue tracker system and what is it used for?

The main things that go into the issue tracker system are suggestions for improvement, they can come from field people, they can come from our internal clinical group. Basically from anybody but those are two main sources for improvements. Defects go in there, software defects, electrical defects are also tracked in there, disposable defects tend to be tracked in a different way, they usually don't go into our Issue Tracker database even though suggestions for improvements for disposables can go into ours. The important parts of the issue tracker are that we have one or more people assigned as having primary responsibility for the issue being resolved. We have a target assignment that says that we'll release for what we targeted for. Issue trackers can sometimes be explicitly unassigned, if we don't know that yet. We have a priority on it. Our first priority is to just give people some indication of what they should be working on if they have, because usually they have multiple issues assigned to them. The kind of things the issue tracker helps me to do, I can get some fairly good indication of how the whole process is working. Weekly we look at things like how many issue trackers got opened against the build. How many that got resolved. Even look at a mix of issues, some issues are testing issues where the software is working fine, it's the test that is the problem. Some more software issues starting obviously electrical, mechanical. Seeing it is a pretty good indication of how the software is, what area of the software are proving the most unreliable. And I also use it to determine for individual developers, how they are doing. Some of the things I look at are how long issue trackers are open for people and obviously, have to judge that with some eye for how complex the issue tracker is. Certainly there are trivial ones that can be closed with few hours of work and there are some that take weeks. Another thing I track is, how often for each developer, the issue trackers are set back. When a developer confirms the issue tracker ideally they've done all the testing prior to QA test and have closed the issue. Of course that doesn't always happen and sometimes the issue trackers are set back, having not fixed the defect or having caused a different defect. So I track that sort of thing. Just giving the people a feeling of you're doing great work, you may be able to do the unit testing yourself before handing it over to. We don't want to be in a situation where we are relying on QA to find every defect. Ideally, we don't generate defects to give something to find, but if we hand in a defect to QA, I see it as a failure on our part. We've lost one layer of protection against the defect getting out to the field. Its an off the shelf software product. The QA group customizes it for our project. Basically there is a standard set of fields in the issue tracker, and then you can add your own fields to it.

What development life cycle do you employ in your project?

I don't know that we have…I mean in terms of naming a life cycle a formal development life cycle. I think that our life cycle builds heavily from the Spiral development model. In the sense that we are dealing relatively short cycles of development, testing and release; we have requirements in there. So I'd say that's probably the best way to describe what we do. I think another important part of it is the cross functional involvements in the life cycle where we try to make sure all the different disciplines are involved throughout the life cycle.

Can you describe the workflow for each release/iteration you define?

The first part of the spiral, I tend to assign one developer. I would use different people each time depending on who has experience doing it. Their job is to go off talk to the clinical people, marketing people, talk to any of the stakeholders that need to get certain features into that release. Then the output from that is a set of software requirements for the system. The kind of tools we use there to track our requirements is just a word document. Because we rely very heavy of screen mockups and those are just generated on simple paper and nothing like some of the tools we're using. But those screen mockups help a lot. I mean in communicating the aspects of the software to non-software people. There are people who you really wouldn't want to show the UML diagram but showing them the screen mock up and describing what happens with the push button helps really well. So that's kind of the front end of the development where the main the outputs from there is everybody's free to say this is what the user interface is supposed to look like, this is the set of requirements we need to do. From there, we'll sit down and determine who on the team will work on different parts of the project. Again, in an ideal world everybody can on anything. In practice, there are some people who are better in a project than others. I do try to mix that around and try to get people when the schedule allows to let people get more experience. So we'll assign then who'll work on that release. We'll generate a set of issue trackers to describe the set of requirements. So we'll get them into the Issue Tracker system and then we'll go through a phase where the software isn't available for QA testing yet. Its just internal amongst a few developers trying to get a basis to our build. When we first tried to get, this is kind of particular for our application but the first thing you have to do when you are working on protocols is to get the data for the protocols to describe. So you have to tell the system things to do. Then you have to set up the procedure, what rates you want to put it on, how long, that kind of things per protocol, a lot of different items. We tend to get that done first. So we use all that data and then get the screens working with that data. Once we're at that point, a lot of people can work in parallel at that point because we got a lot of basic data in place that everybody needs to share. So from that point, things go pretty quickly. Little bit of basic structure developed in protocol. And then long before we hit feature complete we start giving releases to QA. So for the last protocol in particular which I'll experiment where basically QA's spent pretty about two weeks doing pretty much ad-hoc testing. They run it against their test plan trying to see where we had problems in the software. It worked fairly well. I think we did pretty good job identifying where we were [going to] have problems in the software where we will be spending more time to make it better. The disadvantage was that we ended up with a lot of duplicate issues. People didn't know that other people had encountered the same problems. Or didn't know [they] were looking at the same issue. [It shows] slightly different symptoms to the operator. And we ended up with a lot of non-issues because I think we probably did not do a very good job training the testers in terms of what the protocol actually did. So I think we need to improve that next time. Let the testers know 'this is what that protocol does' and 'this is the kind of stuff you would expect to see'. Anyway, after that we moved towards feature complete. We close all the issues that are related to that features in release so we get everything working. And then we go into a phase where software development, most of the people, will move on to the next protocol. There'll probably be back to resolve ongoing issues as QA identifies them. Resolve a lot of times once we get the protocol basically implemented. The protocol people will use it and realize that something they thought they liked really doesn't work out very well and we [misunderstood]. And then we go into a phase where we do final reliability testing of the system. And then hopefully, yeah, it'll be released.

Do you use object-oriented design? UML?

We definitely use UML. And certainly the system tends to be object oriented. We don't really use a formal object oriented design like Rombach or anything like that.

We use UML quite heavily. It tends to be a good short hand way to let everybody know the system is structured. [We use] just Visio. I haven't [used anything else before]. I tend to use it for higher level things. So I tend to use it to show how the larger modules interact and how the larger modules are implemented. I tend not to get too detailed. I think that the particular pieces I like – I like being able to show structure diagrams to show how things are organized. I tend to use the state machine aspect of it pretty heavily. And other people, I speaking as a team in general here. Use cases, some people like them more than others, but I think they can be useful to describe in particular some of the operator actions with the system. Some of the sequence diagrams basically showing the order the events have to happen; the interaction between different tasks helps quite a bit on our system where we have, we probably in our system have about 12 or so major tasks running so the sequence diagrams help show the timing between those tasks and the communication between the tasks pretty well.

What languages do you use for development? Do you use an IDE for assisting in the development? How effective do you think it is?

Almost all development is in C++. There is a very small amount on the driver level that's done in C and assembly. But I bet there are not more than a few of us who are pro basis on C/C++. We also have some grown, I don't think they get up to the level of being languages but…like for instance they set out config files, we describe them in a pseudo language that we can run through a parser that generates all the data structures and handle things like being able to read and write config files, and update parameters and range check parameters and things like that. Have CRC protecting the files and that sort of thing. That's something we wrote in-house. The parser. We probably could have found something that did what we wanted. But we have some level to go that we could reuse back in the older projects. So we had some pretty steep need for that that we needed to fire up. So because of that, because we wanted to be backwards compatible we decided to go with custom build. And the overall development effort wasn't very much.

Most of the people would use SlickEdit. A couple of people use Visual Studio. In order for the VXWorks system that we use quite broadly, for the current version of VXWorks we use the IDE is Tornado. It could be clumsy but not very well suited for large projects at least we've found that out. So tend not to use tornado very much. There are a couple of pieces in the new variety that can be very useful. It does a good job of displaying climbing charts. You can actually show a very nice climbing chart that shows some of low level works that's happening, things like interrupts, message passing, things like when tasks are being blocked from happening. So it actually tends to be very handy in verifying performance issues while we're spending time on things that are more [problematic] than others. Things we get for tracking down otherwise fairly obscure bugs. So that part of Tornado is useful. The debugger is based on essentially the gnu debugger, gdb. Its just a graphical front end for gdb. Its moderately useful for unit testing. The problem we run into is that we're working on a real time system that has substantial watch dog protections, deals with a lot of inner task messaging, locking a task or debugging unless your very careful tends to just crash the system. So we tend not to be able to do a lot of debugging on a live system. But for unit testing, the debuggers usually can step through a particular unit and see what its doing and track down various errors.

What kind of project is this?

Starting at the top we have three main processors in the system. One is what we call a control processor. It handles the sequencing function dials completely to run the protocol. That processor is the first one to be this kind of controlling processor. We have a safety processor. The name is kind of misleading, its certainly not the only processor dealing with safety. There are safety requirements on all the processor. But the main load on the safety processor is to watch what the rest of the system is doing. So the safety processor itself can't do a lot but it has control over… it can shut down power to the system, it can shut down the centrifuge. It can shut down the valves and pumps. So the safety processor has kind of supervisory role of watching everyone else does and shutting it down if its not safe. The third process is the kind of processor that deals with image processing so you can track what's going on in the interface and control the separation of the blood components.

Why do you use so many different IDE's and not just use one that had all the functionality?

Mainly I leave it up to the individual developers. I think its something that people tend to be comfortable with. So I don't really impose a common framework that everybody has to use. Obviously there are something's that have to be common. For instance the technique for building the project, everybody has to use the same thing, I'm not afraid that some people build it in the IDE, some people use make file, some people use batch file, that wouldn't work. We have a very structured make system and then the build process that everyone has to adhere to. So there are some things where absolute standards are required. We can't have it done different ways because… having people build in different ways will end up with people having different outcomes which I don't want to take the time to chase down. But when it comes to what text editor you're [going to] use to generate your software, what debugger you're [going to] use that kind of thing is your personal choice and lot of people use what they are most comfortable with. We do have standards in terms of documentation. In terms of what type of UML diagrams are acceptable and how much detail they have to have. A lot of that is dictated by our requirements. We have coding standards that say what you can and can't use in C++ and sometimes you have to structure things. So I think we fist comment everything useful in terms of individual productivity tools, like I said would have to do with individual debuggers and stuff like that. As long people are producing quality code I really don't care.

How do you manage traceability between requirements to design and design to test? Do you use tools to manage this? If yes, what tools do you use? If no, why?

QA develops a traceability matrix that shows traceability between the requirements, the tests and failure mode analysis, the FMEA. So they are responsible for making sure that we are covered in terms of feature requirements and tests. The testers [have to] make sure that we're meeting all the failure modes.

How do you perform testing – Manual or Automated? If it's automated, what tools do you use? Are they off-the-shelf or custom made? If custom, why was there a need to create a custom tool? – was there no equivalent available on the market? For manual testing, how big is the QA Team? Is this size good compared to your development team size?

We do some of each. I think unit tests tend to be more automated and the systems level testing the QA does is all manual. I know they have some tools to do things like analyze log files and for instance, for some reliability runs they have some automated tools to go to the log files and generate automatically some reports to tell them that they are doing what they are supposed to do. Unit tests - I can ask people to develop. Extremely automated unit tests are like being able to take the unit tests just and run them easily whenever you do a change. In a way its like regression testing. And actually [when] I have some new people joining this project, the first I have them work on is to go through these unit tests, more automate them and expand the coverage of the unit tests.

How do QA and development communicate with each other? How are bugs tracked? How is the fix communicated back to the QA? Do you think this is the best method? Why?

Well there's that weekly meeting that keeps coming up. But we also try to foster very relaxed and informal communication as is needed. And that tends to work pretty well. One thing I like to avoid is to have … I don't like to see QA get stuck not knowing how to test something. So certainly, in that direction we're available to help explain exactly what a requirement means. While I draw the line, I certainly don't want to be in a position to ask the developer to say 'this is how you test it' but I certainly can suggest that, you know, 'here's what the requirement means' and 'here's how you can demonstrate that the requirement is being met'. If we have a couple of choices we talk about in general terms. And it goes in the other direction as well. If they are seeing problems then they can talk to us and sometimes it takes some employees to determine if it is a testing issue or a software issue. And make sure that they don't end up having to spinning their wheels trying to figure things out. And often they'll come to us and say 'testing will be a lot easier if you provided this amount of logging or provided test harnesses into this particular part.' We have a very specific test harness built into the system that allows them to inject various hardware and software events into the system. It works a lot more easily than actually having to – for example, test the centrifugal speed. I don't pretend to stand anywhere near a system and test if the centrifuge is spinning at 400 RPM or 800 RPM. So it tests that condition.

The testers are involved very early on in the development process. So they are a part of the requirements analysis and developing requirements. Actually we generally split the requirements document. The software group is responsible for maintaining the control software requirements. And QA manages the safety software requirements document.

Have you tried any other life cycles other than Spiral modeling?

Not really – no. Not when, you know, the Spiral modeling, after making some small altercations to it, works really well for us.

Are there any gaps in the development process that you think you were not able to address because of a ready made tool, or because of the project/team characteristics? If so, how do you address them?

I mean there've been a lot of things that we have tried to improve and I guess one part of it is described in the Spiral process which is pretty important is the after release we'll have a post review mainly involving QA and software and the project lead. They talk about problems we came up with and production improvement we can make. Some specific things that have come out from the past are we tried to change some of our testing strategies. I guess some particular examples are doing some of the ad-hoc testing up front to see if we can identify very early on in the development life cycle for that release where we might have problems and where we can spend our time. We had discussions over the benefits of doing basically extreme testing, things that would never happen in the field. The classic example that comes up a lot is people will start a procedure, with an 8 ft tall 400 lb patient. And then in the middle of the procedure suddenly change it to a 5 kg infant. And you do end up with problems. Certainly the software cannot handle that smoothly. And occasionally they find problems with the software in general. Often what they find is asking the machine to do something physically impossible and shut it down. Rightly so in most cases. So, we've talked about the trade offs. I think, some of that testing is actually necessary but there's also the tradeoff that spending more testing time on things that aren't actually [going to] happen in the field is more likely to show problems that are happening in the field as well. A lot of times if you find problems with that kind of extreme testing, the other problems say you should fix them, but they are not any problems that anyone in the field is ever [going to] see. So we've talked about the balance testing of that. We identified, just purely on the development side, issues that have helped us enhance our coding standards. We made rules about how different pieces of software have to be developed and written and have to be documented. Because we've identified problems with that. We've changed policies on unit testing because of some of those post reviews. So, basically we do have a feedback mechanism at the end of a release cycle to identify some improvements. I try to identify places in the software that we are having major problems with, that are always bothering people or are hard to work with. And pick usually one area. And basically give people the opportunity to say 'okay, we can throw this small section of software out then we can give or take it but re-factor it pretty heavily. So that we don't have to live forever with legacy problems that we've coded at one time and have reason to be [wrong], I certainly don't want to get in a situation over here where we're doing massive rewrite of the entire project. It means we'll be down for a very long time if nobody will assess how reliable things are. But if we pick one small piece and re-write it then each bit of the software gets better but its just a small enough to get the software back up and running again almost immediately and can assess how well our changes are working.

Are there any external entities that influence your process – e.g. FDA? How do they influence the project? How is this influence handled? Do they affect the design process heavily? How about the testing?

Yes. There are a bunch but the two main ones that influence us are we have an ISO body that verifies that we're meeting the ISO standards. And they basically pick a different area to review. I think they come in once a year, I could be wrong. Seems like I see them once a year. And they tend to concentrate on things like our traceability, how well we're tracing our requirements to actual implementation and testing. They tend to look heavily into our design history file and see how well we are documenting what went into the design, how we made the decisions for why we did things the way we did. So they tend to be that kind of level, not so much the nit-y grit-y software development level but more of the upper level to make sure that we're doing what we say we're doing. And that we're adequately controlling the software development process so that we're getting a reliable and quality product out. The other main people we deal with are the FDA, just because we're based in U.S. And they do some similar things. They also tend to look at field issues and how we're addressing them. Beyond that, other regulatory bodies obviously we have to deal with. Japan is a big one. They tend to be quite different from the FDA. They tend to be a lot slower than the FDA and fairly difficult to work with. We got FDA approval relatively quickly for our process for our plasma exchange. We're basically just starting to tackle the Japanese market and we've probably done maybe six months to a year of grunt work there. And we're anticipating it'll at least another 2 years before we go off on market to sell in Japan.

How formal or informal do you think your development process is? Why? Can it be improved? How so?

I think it's a mix. I think the places that need to be formal, things like failure mode analysis, making sure the system is safe and efficacious, making sure we're meeting market needs, the requirements documentation – I think all that is quite formal. I think the testing plan is quite formal. Some of the more intermediate pieces – we tend to have formal design reviews for an overall subsystem. For instance, we will have a formal design review for the user interface. We'll have a formal design review for the procedural software of one of the protocols in terms of sequencing functional and things like that. We tend not to have detail design analysis for really small parts of the modules. The way we address those instead is to actually be part of the software development standard required by peer code reviews. So at that level rather than having formal design review meeting for some very small part of the module, we want to get the developers to do peer reviews. Safety code is a little bit more formal. Safety code requires a peer review, and it also requires a review by QA. So that actually gets code reviewed twice.

Informal ones are how the developers do unit tests. We require unit tests for certain pieces in the software. But other than writing a template document that basically says what your unit tests [do]. It gives you a way to document your unit tests in a consistent manner among all the different developers. I don't really formally control exactly what the unit tests actually look like so the people who write their own, I have no problem with that. Like I said earlier I tend to encourage people to do automated unit tests but certainly not a requirement. So that tends to be relatively informal. As we go along we formally decided that these are the tasks that are running which are probably [going to] communicate. But some of the more detailed level stuff like detailed conferences, message and screen tasks, a lot of the times those are developed more informally amongst the developers. So somebody working on procedure control, we formally stated that the requirements that the procedure control has to deal with are unlikely to make sense. But the details of what's in that message would be left to the developers, its not really a formal process to set off a requirement that says 'these 27 fields are going to be in the message, here's all the units they are going to be in.' So we tend to allow the details to a be a little less formal.

What other tools do you use to assist your development process? Why do you use them?

Source control, we unfortunately use MKS. Its not perfect. One of the nice things it has going for it is that it keeps everything in a text based file. So, I've never seen a source control system that is completely 100% reliable. But the best thing about MKS is that its all text files so you can usually go in and find out what happened and fix it. Some of them tend to use large monolithic databases which are a lot harder to repair if they break. So it works, its certainly not ideal in any sense but…

I guess the other big piece that we rely upon very heavily is the logging process. The software generates very detailed log files down to, we even log camera images, all the important critical points in the process. We periodically log virtually everything that's going on in the process all the volumes and things like that. So there's whole set of log files and tools that we develop that take various people to generate different kinds of graphs that are all very useful.

C.3 Interview C

Starting off… can you tell me a little about what you do here?

Okay, the software itself is called Communication Manager and its actually the embedded software on the company's PBX switching equipment for mid – to – large customers. There is a variation of it for small branch offices as well. It's a really code base. Its actually survived several generations of products and was kind of imported over to new platforms. Actually the previous generations were all on proprietary hardware. Its been ported over to LINUX platforms, so its not strictly a standard distribution of a LINUX, we made some modifications to that.

Your problem domain? What issues are you trying to address?

Basically, as far as a PBX switch or Personal Branch Exchange is a telephone system that we sell to businesses or government entities so that any sort of business like walking into this building here, you know we have several switches into the basement and everybody's phone is connected into that. The public trunks coming in from quest attach to the PBX and that switch searches the call internally and the switches call externally as well. Of course, there all kinds of different calling features that you can use and different variations of the hardware, different variations of reliability or redundancy that you can buy. All that is essentially, you know I talked a little earlier about the transition from proprietary hardware to Linux and its also in transition from kind of dedicated networks from the Telephony to IP Telephony. Its essentially in the same network as the customer's other data so that's is still just basically making phone calls, calling features, conferencing, hold, transfer, there are about a 150 features that are actually you can get with the company's software; So its fairly rich Telephony application you might say.

Do you directly interact with your customer?

Not, I don't. We build the software in the research and development division of the company. Certainly we are impacted by the customers views that they have or sometimes re-direct the work that we do or influence the work we do but we're not directly talking to the customer.

How big is your team? What are the different roles on your team, e.g. Testers, Developers etc.? What kind of interactions do they have and how do they have it?

We're at 7 now, myself included. We are a little unique. We do some development, we do tools development. So my group actually owns the software bases and the development environment, so something you would commonly referred to as software configuration management, [sort of] the industry term for that. So that's what my group does, we own the software, we own the all the tools the developer use to set up and deliver their code changes. We own the processes that put the software under the media so that you can install it on the switch out in the field. As far as the division of what people do in the group there's, this is not the ideal mix because we currently went through a lay-off so things have [kind of] been thrown askew but, there's 5 people are involved primarily in the building and management of the software for bases and supporting the tools. There are 2 people who are dedicated 100% to the testing the build, the builds of the software we do, [kind of] like a smoke test or a sanity test just to make sure that all the basic functionality is intact and that it is a usable build by the actual software developers or that its good enough to go into the next phase of testing which might be integration test or system verification, it's a broader product test.

Then there are a couple of people who kind of split time between those functions, like one of our engineers for example, works on the tools, the development tools, kind of in her spare time but then her higher priority is to do the sanity testing right now. So it's a mix of build, tool, support testing the loads. Its what we do.

One of the differences we have with other teams is that a lot of times smaller project teams just kind of have the software management that's something the somebody does in their spare time, there might just be another developer on the team that's running the builds do a little informal testing. But this software, this is like 6 billion LOC and 150 developers working in this code base, so you really need a fairly robust set of tools that are centrally managed as well as the software itself. With the scale of stuff we are doing you really [kind of] have to have dedicated people taking care of it.

Can you explain on what these tools help with?

Some of it is you know [kind of] check out check in sort of thing, but with the number of people we have working in the base, you actually make copies of the files you want to edit and we call that a work space so then you make your changes, test the changes. Have an inspection of the changes which can be several different things, which can be like a simple desk check, where somebody is looking at is and checks off or it could have a full blown code inspection, it [kind of] a depends on the nature of the delivery. And then you deliver that code back into the base along with a lot of other people and you might have to merge some of your changes if somebody else was changing the same file. The tools would do a lot of that merging but sometimes it'll get to a certain point where the developer has to do the merging. Once we have all of the deliveries we got a target build but then we keep that build off and rebuild the base and that's the point where it goes to the sanity test to make sure that all the basic functionality has been preserved and that becomes the new base that all the developers work with so my team owns pretty much all the tools involved with the entire process from setting up the work space to all, and all kinds of tools around doing that like C-scope is a tool we use to find problem or find out information about the code that you need to know to do your work. Just kind of, all the surround tools that the developers use in the process. For all the tools that the developers use to set up their workspaces with, that they use to develop their workspace, the tools that they use to inspect the workspace and then deliver. So the only that we are not really heavily involved with is how they test. So basically there are other groups that sort of own tools that they can take their workspace and actually install on it a system and test it so it's a little streamlined that they don't have to follow the actual the full installation or upgrade process that a customer would have to do because that's all they would ever do. So they have got kind of quick and dirty tools for how to do that. So that's the actual part of the development lifecycle that we're not responsible for the environment.

How long does your project last?

Well that depends a lot. We get anything from some minor, we might have some minor bug fixes that we do which could be an hour of effort or a couple to some major initiatives that would take months. And part of the issue there is really that nobody does this as their only task so its kind of like an interrupt driven ride. If you were an integrator but you were also a tools developer then your primary work is building the loads and building the service packs of the schedule that they are on, build on their problems whatever, that's your first job. Whatever time you have left you work on the tool. So it's a little indeterminate you know how long things are [going to] take if somebody were able to sit down and devote all their time to working on something that I'm sure most of the projects would be done in a week or month, maybe if you are working full time on it but you don't really have that luxury.

Do you maintain a timeline? How do you track the timeline such that it is finished within the scheduled time? What do you do if the schedule is not met?

Yeah usually but we don't really have firm deadlines on the stuff you know. Its actually the environment is all there and a whole tool set that we use exist so you know, expect for maybe developing something entirely new which we are doing a little bit of most of it is kind of care and feeding of the environment so there is obviously things that are breakage or high priority changes that we want to get at ASAP but we don't usually … we'll set some informal deadlines around things sometimes but there isn't a lot, there isn't much effort put into trying to schedule things for formally.

Do you communicate with other teams? What communications do you do with the other team? How do you communicate?

Right. That's just mainly email. But then obviously there are face to face meetings for example and there's also an MR system so that's like a change management system. You know most code management, you want to be able to document what change you are going to make and be able to see whether that change was actually done without going to look into the code, so you have the source control where you are actually managing the code, and then you have change control that are like documentation that goes with the source control so we call them MRs for Modification Request so when somebody wants to make a change to the product code, we actually use the same system for the tools. Somebody would write one these modification requests that basically says, here's the problem, here's the system, here's why it needs to be fixed or if its enhancement, here's how we can change it so that it would be better, faster, cheaper, whatever. Then that's actually kind of the documentation of the change that goes through the process with the tool change. So that MR would actually be when we deliver it you know there's the concept of the state of the MR. So it'll go from assigned (which is when somebody is working on it) to submitted (so that they are actually done with the code to kind of ready to be implemented) to approved (when its actually put into production use). So that's kind of an oversimplification but…. And then you have all those different databases where you can go look at an MR by their MR number and see what state its in and see what the resolution was etc. Between people writing the MRs and emails and face to face meetings including an MRRB which is Modification Request Review Board and that's where we have regular meetings where we come together and talk about the MRs that have been written and prioritized which ones we need to work on and get more information about what's actually needed. All those are kind of the ways that we communicate.

Is this MR system off-the-shelf tool?

Well, in a sense. Its actually sort of a legacy tool owned by lucent technologies but obviously used it from way back when it was owned by AT&T and we all used it so its kind of an in house tool but for a couple of spin-offs.

Is that how you gather requirement for new tool features?

Right, we try to. Sometimes the MR would just be written with sort of an abstract idea of what the change needs to be we might some meetings or we might clarify in the MRRB what tools will be needed and then we'll append that into the MR. For tool work, that's the record of what we actually wanted to change or create with the software is the MR.

Does it help you manage the requirement?

It's definitely the quality record of the change so if an ISO auditor wanted to come and wanted to look at our process and say 'How do you know what change to make?' that's what we would talk about – the MR system and how that all works.

What development life cycles do you employ? Why do you use this lifecycle for your team? Does your team size or structure or project needs influence the lifecycle?

It's a little ad-hoc really. You know these tools are as old as the software is really. Our base is a 20 year old code base really, its evolved from something that old. It looks totally different now but for the tools that have kind of grown up around the software. But we're starting to have more of a notion of lifecycle, really what happens is that there are a certain set of people that are happy to write C code and use these all tools which are mainly all UNIX scripts is what they are for the most part. And then there are obviously a lot of people who have worked at other places or have graduated from college more recently or whatever they are used to different you know kind of way of working, so they would rather use the integrated development environment or you know some of the more modern tools which are essentially do the same thing but they are more used to working in a kind of more GUI environment than a command line environment. So you kind of have that. We're kind of working with that now where we are transitioning from little less of focus in the command line, scripting tools and trying to migrate into some of the integrated development environment type tools. So from that stand point there's sort of a notion of a life cycle but we're just kind of making it up as we go along. Good example right now is that we've got certain tools that have got a command line version or UNIX version and they have a web version. It’s a lot easier to implement some changes in the web tool since we got limited resources with some of the tools we’ve actually come to a point where we are getting ready to ‘Okay, you can continue using the command line version for this set of things but we’re not going to enhance it to do the things that the web version is already doing. We’re not [going to] add any enhancements to the command line with the idea eventually that we would get rid of that and just use the web version. So we might have a similar transition where we’re starting to look at eclipse you know and trying to integrate eclipse with our environment and we’ve picked out the first part of the development process that’s come with the eclipse plug-ins - that set of things. That part of the development work. And if that really got a lot of traction and got a lot of use we would probably do a similar thing but on a bigger scale and then kind of stop maintaining the legacy tools that do the same things that eclipse does, except where there’s some reuse. So that’s really our only concept of life cycle; kind of make it up as you go along.

How do you manage traceability between requirements to design and design to test? Do you use tools to manage this? If yes, what tools do you use? If no, why?

Right, there are fairly rigorous requirements for the product MRs which use the same process. And you can do that kind of forward and backward traceability in that process but we’re not really as rigorous with the tools. A lot of the times we write the tool and then we write the MR to deliver the tool having never really written down in your requirements. The tool developers are fairly experienced and they what is needed and they iterate on it until they have what they think it needs be and then everybody says that ‘Yeah, that’s what we needed’. So its not really as rigorous as the actual product code itself.

Do you think it needs to be?

The people that work on the tools have a lot of experience and even our newer developers get a lot of good mentoring and the people are doing inspection and helping them test things so the team does a pretty good job of you know making sure that we’re really doing things correctly and with quality and it seems to work pretty well. So yeah, more rigorous requirements are not really necessary in this type of work.

There are some exceptions to that as far as the requirement. We’ve actually a fairly large set of enhancements that we did starting in April that we’re still not done with but those are some that we actually did sit down with the development and over the course of few meetings we developed a couple of page description of how the tools should work and that was more along the lines of requirements, so there are some exceptions. And we’ll normally sit down with a developer to kind of figure out ‘You want this enhancement, let’s sit down and figure out what you really want and we’ll append that into the MR and that’s more of the requirements. It’s not real formal but we really do talk about what's expected put that on paper before we start working on stuff usually.

What languages do you use for development? Do you use an IDE for assisting in the development? How effective do you think it is?

Most of it will kind of be in JAVA, and there's one other language that's fairly predominant with Eclipse that one of the other guy who's working on it is more comfortable in so we're [going to] end up with a mix of the two. It escapes me right now about the other language.

You have all these different tool sets and different processes, can you at a high level can you give an overview of the tools and what the uses are?

There are a dozens of tools that do this or that little minor task around it but there are several major ones. They all begin with PJ which means project tools, that was kind of a legacy name for our tools.

PJSetup: And you use that to set up workspaces, along with other tools that you can look at to do some searches on the base to figure out the dependencies of what you need to change, what you need to pull in

PJInspect: Once you set up the workspace then you've built it or tested it and sort of iterated on that to where you think its pretty much where it needs to be and to start doing some inspections on it so PJInspect is the next tool.

PJMoveWorkspace: So once you inspected the workspace and all you inspectors agreed that you can deliver, there's actually an intermediate tool called PJEndsWrap. Its where one of the inspectors has to approve the workspace for delivery.

PJReportsBase: Which you actually use to deliver your code changes into the delivery area, along with everybody else's changes.

And obviously we use object generation or obs in the builds so that's the kind of the high level tools that if you watch the workspace go through the process, those will be the ones that really kind of define the high level of the development process. But really within any one of those steps there could be a number of underlying tools that the developer would use during that phase. I've never actually worked in that myself, I couldn't describe to you too well what they are but …

When does your build step happen?

At the end of the whole process and that's the build of the entire base. Obviously the developer has built their changes as worked on and put the changes together.

What tools do you use for your build management?

That’s a whole other set of tools. We call them the integration tools and that’s essentially, development owns the make files in the base but we got a whole other stuff of tools that really manage the set up of the delivery area and merging of the workspaces when people go to deliver then building the entire base when all the deliveries have been made and there’s actually several different software bases there’s a whole process for the daily builds, there’s bi-weekly builds, so there’s actually several different software bases that the changes would go through in order to accommodate that. There’s an in process daily base which is kind of the one that the developers deliver to, kind of iterate on their changes, that base has to build and pass a set of tests and when it does pass those tests, it becomes the stable daily base. So people who don’t want to be kind of on the bleeding edge will work on that base. And every two weeks we take the last stable daily base and grade it with some other changes that we get from some other parts of the project that aren’t actually, the sources, aren’t actually in our base but they’ll deliver their RPMs or OBJS in our base and we’ll [want to] grade them together with it. Then when we do the sanity test, then we either hand the base, the software off to the further testing functions or that’s [kind of] rather defines the next cycle for the daily builds, then the next set of daily builds will start from that bi-weekly builds. So it’s a fairly complex process doing all that. So there’s dozen’s of underlying tools that build the offer when we get these little components from other parts of the projects. There are 5 bases we build the changes in. There’s a base to run clock works in, for example, for you to find syntax problem and stuff like that. There are a lot of layers and tools that have quite a bit of dependency between them to do all that stuff. There’s several hundred tools easily if we counted them all up.

Do you use MR System to manage all of these?

Pretty much, yeah, when we make changes to them we require a tool’s MR, even if it is written after the fact.

How do you manage the documentation?

Some of the tools are not really documented too well and that was actually one of the things that we had one of our engineers doing was reading the tools, seeing what they and writing a little summary of the tools. She actually spent some time doing that. That was more towards the integration side where really developers don’t need to understand the tool or what they are doing, while its really only our group that’s even needs to know anything about him and if they all work, you don’t even need to know what they are doing. When they don’t work [kind of] understand what they are doing so that you can go in and understand what the problem is. So that part of the tool isn’t as well documented but all the tools that the developers use, the PJ tools have, they have man pages so that you can pull up some online information about ‘Here’s what the tool is, here are the arguments, here’s what it does and here’s what the different arguments do’. We have a website that has similar kinds of information that people can [kind of] go in a look at what all the tools are and tool to do a particular thing.

Those are the primary things, and then development has their process document. They call it the project procedures notebook and it [kind of] starts at the very beginning with here's all the things you need to be able to do when you come in a new developer, all the login information and the stuff you're [going to] get and from that point to how do you set up a workspace to how do you do this stuff. So they've got some references to the tools and all that documentation as well. So that's [kind of] the major things that people have access to. And there's also obviously other websites and information from here and there.

Can you describe the workflow from when you get a MR request?

That's [going to] be up to the individual tool developer. There's not really a rigid requirement. Sometimes if they are just modifying an existing tool then they are more limited in what they can do than when they are writing a new tool from scratch. If you are [going to] fix a shell script, you're [going to] write a UNIX shell. You're not [going to] off and write a PHP tool or something. So it's a little bit dictated by what the actual change is but then developers kind of get comfortable with something and that's kind of what they use so you know, got a couple of people who most do shell/UNIX scripts and I've got other people who have got a broader skills, so they'll do a little more PHP or JAVA. And really if got some firm requirements from the development about what's really needed you know there are certain things that really lend themselves to what tools, they [kind of] dictate what you need. So its [kind of] half what's needed and half developer preference. What established languages that we work in are …

Who does integrated testing in the team when a change is made?

There's obviously a lot of interactions between the tools, a lot of interdependency between them, so usually the developer will depending on how complex it is, if its just a couple of tools that are dependent then he will test all the scenarios that he can think off himself. One of the other developers would maybe do a code review with them and they would agree that that was sufficient. But if its 2 or 4 tools that are interacting, the person that's writing the code would probably do their testing but then they would usually ask somebody else to test some other scenarios, maybe they would them to focus on some other things that maybe they didn't do. Just having 2 sets of eyes on it, 2 people come at it from a slightly different direction it is beneficial. They'll also do the code inspection together, that's also there.

How do these changes get back to the person who reported the defect?

When we are developing the tools we are developing them off in the side like in a private are. The actual production tools are kept in a bin, an official area where official tools reside. They are owned by a particular login which only we have permission to go in write the tools or change them. And some of the tools are actually under source control, we use CVS for that. They are in kind of a production file structure, that's where the official tools are.

Person requesting the changes only have access to the downloads, so once we are happy and we have tested and inspected the change, we'll write it to that area. And when they set up their environment and run their tool, they are running it from that area so they have no choice. And usually when we make most of the changes, we send email that lets the developer know which tool we changed and what the changes are. That's kind of their notification that something is [going to] work different.

Did you ever have issues where you didn't get a change in time?

We have a lot of scenarios where we actually had to… maybe if we find a bug in the tool something that isn't co-operating somehow, they've [got to] get their code in, their deadlines for all the builds, sometimes we'll actually modify a tool as kind of a work around and we'll tell the person to use this version and that will help get them around whatever their problem is and then that version goes away and we'll hopefully fix the problem in the real version of the tool for the environment and that happens with some degree of irregularity. As far as the other scenario, if there's a developer who's really kind of a driver of requesting a certain tool or change then we will usually get them involved. A lot of the changes are [kind of] straight forward, we all agree that its … A lot time we will prototype the tool with the developer, then they'll try it, give us feedback and then we'll iterate on it a little bit until they say yes, that's what we want.

Did you have to address problems in your development process by building tools or getting them off-the-shelf?

Yeah, again that's mainly kind of at a lower level so most of the primary tools that the people use are these legacy tools that have been around forever. An example might be when we had more international development, it was easier and less latency, networking issues for them to something with a web based tool rather than logging into the environment here and using UNIX scripts, so that's [kind of] what drove some of the web tools to get created. So its kind of what might be solving the problem. All these developers used to be here and now there are some in India, some in Germany, and some everywhere else so then the needs of the committee change overtime and the make up of the community changes and that is driven the way we develop the tools or change. Driven us to develop different tools for different problems. We mainly do what development needs, when their needs change that kind of drives us to do things differently.

What characteristics do you need from your development process – Traceability, Reliability, Verification, and Usability? How do you achieve these objectives?

We don't really measure the tools in a lot of those ways. Obviously we want them to be available because we got developers working pretty much the clock. There are people in Germany, there are people in India, Israel that have to deliver in their stuff and obviously we have to go home and sleep so we [want to] know that the environment is up and working most of the time. But a lot of that is fairly built into the tool because it’s a legacy set of tools that have been around and have demonstrated to be fairly reliable. They are sort of typical things that would happen that you kind of know the work around when some certain scenario happens that we are aware of. And some of those might be because there are some bugs that are too hard to get to the root of but we know how to get around it, so that kind of thing.

Are there any external entities that influence your process – e.g. FDA? How do they influence the project? How is this influence handled? Do they affect the design process heavily? How about the testing?

ISO. Well we’ve gone through some of the audits periodically. That’s really the only one that’ll come and have audits and we actually have to modify the process to make sure we comply with if they find issues. There are other capability audits that we have gone through because management wanted us to sort of assess the capability. They are kind of external benchmarking things and we did that within the last couple of years, it was one with different levels and they came back with what they though what level we will be at. And we were actually fairly at an advanced level than someone who they normally just started working with. We kind of a real focus on process all along. It’s the software capability model.

It didn’t really influence the process. They sort of validated what we thought where we were anyway. And since it was really complying with something so it was kind of pushed on and we didn’t really do this thinking we were going to learn a lot of how to make the process better. It was like ‘you have to do this because we want everybody in the company to do this’ so we say ok and we see what we can take away from it. We actually have some research people and there are guys involved in software research and they write papers every year and kind of keep track of fate of software development within company. They are familiar with all the latest trends and they’ll sort of look at Avaya capabilities and what other projects are doing and determine best practices of who is doing what. That’s more of an internal benchmark that probably gets looked at with more regularity then the external audits. For ISO, you just want to maintain certification of what they’re [going to] expect and what they are [going to] look for and make sure that everything is in compliance with it when it comes.

How often are the PJ tools used? And by whom?

Daily. By communication manager and software developers and by other people who are closely associated with that. So there’s probably about 100-150 people, most of them are the company associates, there are contractors like WipPro in India, they work in the base and use these tools. They work in the contract basis. The major tools are used pretty much continuously.

What other tools do you use to assist your development process? Why do you use them?

There are ASM compilers, couple of versions of that we use so there are lot of open source code, or vendor code which are actually part of our code base so its actually whole process around that the developers have to follow in their management when they want to use some open source, some vendor code. So the lawyers have got to review you know the license for that and make sure that its something that's being used appropriately. If we're [going to] use it, its not [going to] violate the license or leave the company open to some kind of litigation. So there's that, so we don't really write all the code we obviously grab a lot of stuff from the industry. It runs on a LINUX platform and we use RedHat and we actually use their supported versions of LINUX. So we got a lot of the regular distribution of LINUX and we also got things that we have modified. But within the tool and the development environment, and not the product code, there's lots of stuff that just heard of the UNIX/LINUX distribution that we use, SSH or RC distribution that we use with some of our tools. There's a whole set of tools called the EXP tools that we use in the automation that I think are owned by LINUX as well, so there are just layers of the things like that, that we need as part of the environment. And a lot of people don't even know that this stuff exists or that they are using it. They are tools that the other tools depend on and unless you are a tool developer they are something you wouldn't even know.

How do you perform testing – Manual or Automated? If it's automated, what tools do you use? Are they off-the-shelf or custom made? If custom, why was there a need to create a custom tool? – was there no equivalent available on the market? For manual testing, how big is the QA Team? Is this size good compared to your development team size?

They are mainly automated; that's whole other area of tools that we got internally written so we got tools as well as something called Tarts Prairie which is a test scripting tool so you can basically set up a calls and exercise features within it, so it's a way to automate things that you otherwise have to sit there and punch a lot of buttons on the things to do. And that's what the basis of both the daily and sanity testing is these automated scripts that just exercise a lot of functionality of the product, the code itself. We actually have lab models that are actual product switches the software is running on that these test scripts are driving that casts against the software. The daily tests are fully automated so it runs overnight and somebody comes in and looks at the results. If they see failure they have to investigate them and figure out whether it was really a failure or did the script just fail for some reason and then sanity test is more real time; so maybe there's piece of it that's automated and somebody is just sitting there and watching it and if it stops they have to figure out what went wrong; is it a script problem or is it really problem in the software.

How often are these sanity tests run?

Within a particular release about every 2 weeks to 1 month and it just depends on the schedule for that release. So a release that's in development, so for example we're [going to] put out a release next May for example, that release is sort of a full swing of the development cycle and we build that every two weeks. After it launches, and we are selling it to customers then we build that every month, to just kind of put the bug fixes in, there might be a minor release put out to add some of those features or functionality, but we also build service packs for the releases, but that's a whole another load line really. But obviously are related.

Why all these PB tools developed and off-the-shelf software were was not used?

I don't really know the history of why they developed them and didn't use an off-the-shelf environment but all the stuff kind of originated in Bell Labs and there's a fair amount of not invented here in the bell labs. Obviously with UNIX scripting, it's fairly flexible and you can make fairly functional tools without a lot of, you know, you don't have to know a lot of different languages you can do a lot them in the scripting, so I think it was flexible. There was a lot of rigorous process obviously and a lot of processes enforced in the tools. If you go to a Vendor and say 'here's the process we [want to] have and we want you to modify your tools to have our process in it, that gets extremely expensive. And by the time they come out with a new version of their tool, you've got a custom version of that has all your changes in it then you're [going to] pay to move your changes to their next release. Its pretty expensive to do that. So I think that's why they went with home grown tools to begin with. It really allowed that flexibility to customize the tool to the process that you wanted to do.

C.4 Interview D

Can you tell me a little about what you do here at your company?

So we develop project management software, most specifically Agile Project Management software. So we have a software as a service that allows development teams to track requirements, tasks, test cases and defects, along with integrations to probably about 15 or 20 other 3rd party tools that would help along a software life cycle. Company C also offers services as well, some consultant services.

How many different projects do you have here?

Totally depends on how you slice and dice it. There's really one main product, its broken down into editions. If you're a single team you can get a small edition, a medium team, a medium edition, a large team, a large edition. That's the majority of the work, but then there are these 3rd party integrations, some are larger than others. Those probably take maybe 15 - 20% of our engineering effort. So they are pretty small efforts in the greater scheme of things across.

Do you directly interact with your customers?

Absolutely. I do more than others as a project manager, its part of my job description.

What kind of interaction? Requirements gathering etc...

Everything. We're a small company, so I help gather best practices, I help gather requirements, I'll walk and reach out to people and ask them 'Hey, we came up with this idea, what do you think?' So yeah, we do a lot of interaction with our customers. I am on phone calls all the time with my customers.

How big is your team?

Depends on what you call my team. I have nobody reporting to me. Our dev team is about 20-25 total people. That includes testers, documenters, management, product management, developers. We are fastly growing, we're doubling every year.

You have different areas that assist in Agile development process, can you explain a normal workflow where Company C's software would play a part in a development process?

So you [want to] know how we use our tool, and how our customers use our tool? Again, as I mentioned, we do requirements management, in the form of stories, test case management, defect management and task management. So basically what happens is the product managers put together a back log of stories, they rank them and say 'here are the things I care about in rank order'. Developers will then pull from that back log. The developers, testers, product managers, documentation will then get together and define acceptance test for those stories using sometimes test cases in Company C's software. They'll also then create tasks for the stories and assign them to an iteration and commit to that story. Then they will develop that by completing tasks and writing automated test cases. And sometimes during development they'll log defects against them and sometimes defects will come in after the fact. So, very high level that's really how we run our process and how our Product works as a tool.

How many customers do you have?

A few 100 ... 300 customers maybe. And probably around 10,000 subscribers. So customers is companies and subscribers is individual users.

How long does your project last? How long does an iteration's release cycle?

The interesting this with our product is that we deliver in a 7 to 8 weeks cycle, so we have software deliveries every 7 to 8 weeks. So any "project" really never lasts any longer than that generally. Although some if they're very very difficult, will require a lot of thinking, we'll start it before a release starts or in very few cases we'll expand a release. We'll have a team working on one task - one story or one group of stories - for more than one release. But as you hear, that happens very seldom, maybe once or twice in 3 years.

How do you manage the timeline for these release cycles such that they get completed on time?

Well timeline is interesting because Agile development is very different right? When you look at waterfall it talks about the iron triangle of requirements, time, cost and we're really fixing the time and the cost, or the time and the people and we're changing the scope. So basically we set a release and that release date never slips. So when we have release on December 15th, like we did couple of days ago, that's the day we hit and we'll modify what scope goes in there, so timeline in Agile is a very odd concept. We talk more about timeboxes. And these timeboxes are important, its important to get something out in each timebox. And we actually in the tool, and in here have two forms of timebox. There's one as an iteration which is the short internal deliverable normally. And then we have the release which is a longer although 7 or 8 weeks is pretty short in a software business. Its in there releases that are a little longer external deliverable.

If for a particular timebox you have certain tasks set up and if for some reason you don't manage to finish that scope, do you descope it?

Certainly, but since we have a small team and short commitments, it doesn't happen very often. Very seldom. Its happened - once or twice in 3 years. And that's one piece of software that doesn't get fully out. Plus our commitments are pretty general. Right, we don't give somebody a 50 page requirements document saying here's what we are doing. We're [going to] say 'okay, we're [going to] implement drag and drop ranking' or indoor queries on custom filters so we don't go into huge detail.

How do you do inter-team communication?

It depends. A lot of it is through spoken word, right communication. Our teams are all co-located so they all work in the same general vicinity. So our teams are maybe 5-7 people, and a product manager, tester, doc, developers and we all sit together and we have meetings everyday. We have a stand up meeting so that's the way we do a lot of our communication. But some of it is done through the requirements or stories as well as all the tests and the test automation so the communication of what the tool should do is written in test automation.

Can you talk a little bit about your requirements and the stories that you mentioned till now. How do you define these stories and mentioned you track your requirements through test automation, can you explain more on that?

So the stories are usually again, very general. You know I mentioned one thing about 'drag and drop ranking'. When we write a story its more that 'drag and drop ranking' but certainly no more that 10-20 words. And the rest of it is [sort of] talked about within the group. What we do though is that depending on the content of the story we'll the project owner or the project manager will start to talk to the team about that before we get any input. As the time comes to make a commitment and start developing it, all of us will get in one room - product owners, testers, developers, documentation people - and we'll work through what exactly that means in the context of the application and define a set of, a list of 5, 10, 20, 50 acceptance criteria that if you drag some things here it does this, if you drag this way it doesn't do something else and we'll go through and list all those acceptance criteria and then we'll try to automate as many of those as we can. When that test passes that means a story is done or accepted.

Is there any Test Driven Development going on around here?

A little bit, I call it TDD - Test During Development. So we actually have our developers and testers - those are 2 different people usually so they actually will work in parallel.

What's the ratio between the testers and developers?

Probably about 3:2 - 3 developers to 2 testers, so 1.5:1. And our testers are really code writers, so they are developers. They are writing Ruby code.

You mentioned that you start in parallel, so when exactly does the code get to the testers to start the testing?

Depends, a lot of the times what'll happen is earlier on the developers and testers work in parallel. The developers will start writing code and the testers will start writing the infrastructure to test that code if it doesn't exist and it'll come together overtime. And again, since we work in a 2 week cycle any of that code is [going to] start being real very shortly within a couple of days. The testers and developers sit right next to each other and work together and come up with a plan, its not always the same plan but again since these teams are small the people are co-located, I think its reasonably easier to come up with a plan that doesn't cause anybody to be lost for too long.

When does the documentation come in?

Around the same time. It depends, for some of the functionality its really easy to explain. It varies, for some features we do really good mockups, the documenter can really look at it and say 'okay, I understand how that works, how its going to look; I can document it'. Other times the feature is so easy or something you've done before which means that it'll be easy to document. Sometimes we have to wait until they see it so it all depends. Certainly there are often issues that require the documenter to wait in cases but we're working on two week cycles actually helps even things out. We try as best we can to get all of that stuff done within the same iteration but sometimes documentation slips a little bit.

So you yourself use Agile Development for your project?

Yes, absolutely. I think we'd be crazy if we didn't, who'd buy our product, if we didn't use our product?

So did you have to adjust the lifecycle to address your team/project needs?

There is no Agile process, no you do 7 things this exact way and you're done, so we live in a gray world. There are certainly some people who do Agile but that probably think they are but aren't but that's all opinion. So there are core set of 4- 5 things in Agile that I think should generally do and we certainly do all of those, some of them we do really well. Others, its a little bit harder to do.

Can you tell me more about what they are?

Some of those things are co-located teams, you [got to] have product managers, developers, testers QA, doc all sitting together and working together on the same thing in the same team with the same goal. The other is timebox development, so the timebox is more important than the functionality or the team resources, so its always ensure that your iteration is short. The other ones are continuous integration, so you have to have a build system and be able to make sure that your code always compiles. Another focus is on test automation, and unit testing. So, unit tests or automated tests is always way better than manual tests. What else, defining requirements as stories, instead of going through a huge requirements process. That's pretty good. Collaboration between all those different team members, so yeah, the five basic things and we certainly do all of those.

You mentioned continuous integration and trying to build the project? How often does this integration happen?

Always. So as soon as a check-in is made the integration server runs the automated tests and sends out a report. Its all automated. We have continuous builds that happen on every check-in, we've nightly builds as well that go through a wide variety of tests and other performance metrices like code coverage, and code consistency and other tools like PMV and find bugs and other things.

What tools do you use to facilitate all these process?

We use Hudson, open source continuous integration server. Its really very similar to things like cruise control, a little more flexible for us. We certainly use JUnit for unit tests, we also use something called Celinium which is a browser driver so obviously we have a rich web app, web 2.0 app, a lot of Ajax, a lot of java script. If we don't test that we'll kill ourselves because there is a lot of functionality there. We actually invest the time in a set of Ruby code that basically maps to our application, so if you have a button in the application or a tab there's an object in our Ruby code and then we can write code that calls that code and then goes through Celinium to actually drive the browser and click on things so we can run on Firefox, IE, using those technologies. That's a huge help for us and its fun to watch. We're actually working this week on parallellizing that sort of testing.

In the dev process itself, we certainly have tools we written in house to do load testing, we use things like J-probe to look at memory usage and profiling. J-meter we use, obviously you can tell we're an open source java stack. I think that's about it, that's a good start at least.

We use our software to run our whole process, we define releases in iterations in our software, we define stories and test cases and defects. When I started I thought it would be hard to have a defect in the defect management system or writing a test case about the test case management system but its not as hard as you think. Its sort of recursive.

How do you proceed when you find a bug in the software? What process do you use to handle that bug?

Some bugs are found internally and we decide when to fix those within the iteration or release and we try to keep that bug count very low. We probably have about 20-30 open bugs right now, maybe. When we close a defect we create a test case to make sure that defect doesn't occur. So that's our internal process. When we find something internally, a lot of times that will happen on the dev branch code so it doesn't really matter those certainly don't count cause they are found internally before something is released.

If we find something after we release, usually it comes through our support organization, our QA managers basically triages those and looks at them to say 'yes, this is a bug'. Sometimes a product owners get involved, cause there is a fine line between defects and feature requests, and then we try to fix those as soon as possible. Since we are a hosted app, our bug kill rates are incredible, we track average close time for defects and P1 defects just a couple of days. So an average P1 defect probably stays open for 3 days, its in production that Saturday morning if we fixed it during the week. That's actually one of the cool things about doing fast because if we find a bug, we can fix it for Saturday and we can fix it for everybody. Sometimes we do it during the week too, though we have it scheduled for Saturday morning. We use our own software to manage these bugs, what else would we need?

If there is a defect against the defect management system, then we file a defect against the defect management system in the defect management system. Our software keeps track of the defects, and whose submitting it and what status is it, whose the owner and what description it is and the screenshots, and all those other things.

Another interesting thing that we do here at our company is that we have those CRM systems, customer relationship management which is sales force. So if a case comes in with a defect, we can link that case with the sales force to the bug or defect in the software and then we can actually push out notifications back out through the sales force to customers and say these defects are going to be fixed on Saturdays. We do the same things for feature requests. So when a feature request comes in through our CRM system, we then link to a story in the software and then push out notifications so just last week we pushed out maybe 50 notifications to people that said 'Hey your feature has been implemented'. Its pretty handy and customers like it too.

Do you do all automated testing?

No, we do as much automated testing as we can. There are certain things that automated testing will never ever find. It only finds what you program it to and human brain is a lot smarter than the code thats running on your server, so, we'll do exploratory testing and play little games like persona and roles and other things with the team and with the rest of the company, [sort of] to tease out those bugs. We may not find a lot the boring stuff like make sure the column sorts when you sort it and those are all automated but other things are really just harder to catch and other things you won't really automate like 'the color of this chart is wrong, or doesn't quite match' or if there is some usability issue. I mean, there are certain things that you just [got to] manually test for. And everybody does this not just the testing team.

Your teams are co-located so is that how you communicate between teams?

Yes, usually but sometimes we do so through our software as well. You put notes in the defects, 'Hey here's how you fix the defects' so when the support person notifies the customers they can say 'here's what we did' and so we use our software to do that.

Can you talk more about your test management system?

Sure. So, its interesting because before what we called SLM 2.0, before Software Lifecycle Management system 1.0, really what people did was took a task management system or Microsoft project or something, and they took ReqPro as a requirements management tool or Bugzilla or test ReqPro or ClearQuest as a defect management system and then they took test management system like Quality Center or something and they bolted all those things together and they don't go together very well. And so what our company did was basically took in all those things and put it into one system. And it has nice traceability between all those objects, so we know if there's a defect related to the story or a test case to a defect or a task to a story.

So, because we have that traceability is we can create the story and connect a bunch of test cases to it and key the results of those test cases. Interesting that is the area of the tool that I think is the weakest of those 4 for us in the product map. We've also done some work already into integrating build results. So, a lot of your unit tests are run in the continuous integration server and we get the results of the unit tests in our integrated build system. So we do a little bit of both of those things, so we integrate the build system and to get a view of what's going on in the continuous integration server and so we can run from there and see how many times you're passing or failing etc.

What characteristics do you need from your development process – Traceability, Reliability, Verification, and Usability? How do you achieve these objectives?

Those things are all interesting, those are to me all over the map. Usability is really got to be in the product, it also has to be special, we actually hired a person dedicated to usability. That's her sole reason to be here to talk to customers, how people use the tool, how the tool is designed, what the analysis of who uses what screens, what's the most important thing to change, so we certainly focus on that a fair amount and because we are a web app and because we want a rich client GUI, we have some issues that we have to work around and to be able deal with Ajax and Java script is much harder to deal with in certain cases. We certainly look at the Quality, Reliability from your perspective and I [sort of] see it as - reliability to me is the continuous integration. I know that I've got this safety net of test cases that I run all the time, so I can really see what's happening in my code base, what my code coverage etc.

What kind of tools do you use for code coverage? Also, anything else you use during your testing phase?

Clover. We use JUnit, but Clover and JUnit as totally different tools. Clover is a tool that gives you some insight into your code and how complex it is. Our TMV in that finds bugs in the code that might be problematic. Those are things that we want as well to look at.

What are the different languages do you use here?

Mostly Java and Ruby. Ruby's object model is extremely flexible and so we can extend the language to read task, tab.click. There's also a Method missing class, so if there is a method missing in the class, it calls this Method Missing method before calling an exception. Its very flexible.

Do you use UML?

No way. So Agile talks about minimizing the work necessary to get product out the door. In 98% of the cases I think UML is an overkill. I think its useless in many cases. Not in all cases, in 5% of cases, you design something really complex that you want get ahead of some functionality and a UML diagram makes sense but we never ever require that before we actually write code. We do our design on whiteboard. Take a snapshot and attach that picture to a story and say here's our diagram and maybe UML.

Sometimes I think UML might be very useful, but in many companies I've seen they say you have to have a seq diagram like that, activity diagram, UML diagram, we [got to] have all these things before you start writing code and people get bogged down in documentation process instead of writing code.

We don't need to represent any classes, or interactions before you write the code. It might be useful afterwards when you are teaching someone about your code. If you are in the code all the time and you are using Design Patterns that are common and architectures that are common, you don't really have to write all that stuff. We're not a business that has incentive to do documentation. My goal as a product owner or product manager is to deliver code and not the documentation about the code. If I can deliver the code with a lot less documentation and still deliver a quality product I want it. I don't think that's always the case and I was in another industry that's not [going to] work. But interestingly on the other hand, using a tool like our software you have, you know every task, every test case, every story, every defect, and interestingly because we have a tool like that and every object has a revision history and when user logs in, I know every change in every artifact all the time and so while we have a pretty light weight tool its also interesting that we know every change that happens in that sort of artifact model of the things that we care about that define our software process.

Are there any external entities that influence your process – e.g. FDA? How do they influence the project? How about your customers?

Not really. The certainly want input into the road map and input into the features and how features change but that's not the process. Very slightly related might be we have that full service API and so we keep back log of its versions for compatibility. They could care less of our internal process.

What other tools that you use?

IDE, Eclipse, IntelliJ - mostly IntelliJ here. Subversion for Source code control. Fit and fitness for some types of test cases, Cactus for some types of test cases, HTTP Unit, Xml Unit. All those tools are good for certain things, fit and fitness are really good for tabular data like our security model is all tested with Fit and Fitness because it really nicely lays out tabular information but its terrible for UI testing.

How do you train new developers?

A new hire would be paired with a experienced person and they'll work for some weeks together, do pair programming in a mentor program. To figure things out. Here's where you should start and where you should fix the bugs.

Lets talk more about the best practices that you mentioned earlier...

To me its interesting how overtime best practices change. 20 years ago...30 years ago it would have been a best practice to use source code control, and now you're stupid if you don't use them - you're crazy. Now it’s a best practice to have a continuous integration server. You know 10 years ago it would have a best practice to have a build system like Ant. And so this bar continues to rise and you have this infrastructure that's very consistent across different companies and different applications, right, everyone can benefit from continuous integration or source code control or shared build systems - so that is continuing to rise. It’s interesting to watch that; to see what’s the next thing coming up.

You mentioned that you talked to your customers about the different best practices as well.

Yes.

Is that kind of like implementation of these best practices at your customer sites or ...

We don't do as much of that as I think we should as a company. We talk more about management process instead of engineering process; management process I mean like defining requirements, getting the teams to work and collaborate together. We have a services group that does that. We don't really have a practice; we don't have a team that really focuses on best practices. So we're in control of branching and goals and test automation and all those things we talked about as a company.

Is that something you would like to expand on in the future?

It’s possible.

You mentioned that you have different kind of editions geared towards different team sizes. What do you mean by small, medium and large sized teams? And what kind of differences does Rally software associate with each team size? Why?

Small is less than 10, Medium is 10-50, Large is greater than 50. For smaller teams, there are less features. Small teams cannot have multiple projects, or programs that aggregate work across multiple releases, etc.

You mentioned that sometimes you do mockups that you can just show to the documentation people. How do you create these mockups? Are they attached to your stories during your iteration for traceability?

Mockups can be anything from a screenshot of a whiteboard, to a nice Photoshop/Visio type document, to an interactive prototype with customer’s data. Yes, we attach them to stories.

C.5 Interview E

Starting off… can you tell me more about what you do? Your problem domain? What issues are you trying to address?

I am a QA Engineer for our 3D modeling and visualization product. So my role involves coming up with the test systems, either refining test systems or developing test systems that can effectively test the software applications.

Are these automated tests?

Ideally. That’s certainly the goal. Just the nature of automation though and the scope of 3D graphics a lot of the stuff doesn’t blend itself to automation; so we try to automate as much as we can and then cover the rest areas through traditional methods.

What do you mean traditional methods?

More hands on black box, actual human interaction type thing.
How big is your team? What are the different roles on your team, e.g. Testers, Developers etc.? What kind of interactions do they have and how do they have it?

QA Team is 7 people for this product.

Development team seems to be growing daily so it’s hard to keep count. I’m guessing around 20.

Yeah, we actually have a lot of interactions. The QA team sits directly with the development team and a lot of discussion about new features and ways to implement certain features, QA will actually weigh in on those as far as the risk of deciding one implementation over the other. As far as QA time or what that might affect or how that might affect users. Also at times development will come directly to QA and say “I’m working in this area of code, I can really use some test automation to make sure I’m not totally breaking every other part of the code while I’m doing this.”

How do you facilitate these interactions?

Its all ad-hoc so just sitting directly in same areas discussions will come up as needed basis. There is a weekly development meeting for the product which all of development and QA sit in on and discuss weekly topics but as far as throughout the week, if something comes up its easy enough to just go and walk over to the other persons desk and start discussing. The proximity of where we sit and somebody else on the team can also overhear that discussion and way in their opinion, if they hear something they disagree with, they’ll stand up and talk.

What is the ratio between the developers and testers when they are working on a feature?

Given the ratio of QA to development there are no real assignments directly with, say, with QA to features. It may be you know on the scope of the feature and who ever are available when that gets implemented. The developer may say “Okay I’m [going to] have something for you to look at on this date” and depending on what the QA team is at that point there may be two people available and the actual scope of that feature may require two people or may just require one person which gets, you know, one person will say “Okay, I can look at that now while the other person has plenty of other things to work on.

What development life cycles do you employ? Why do you use this lifecycle for your team? Does your team size or structure or project needs influence the lifecycle?

Nothing totally rigorous as far as any of the development philosophies out there, I guess. If it models anything, I guess it would model a waterfall method. But we try to break that down more towards Agile to where we will have smaller iterations per say within that development period. The nature of our product being graphics and being desktop application, the development lifecycle for adding new features between release and release can take several months up to a year even in some cases. There’s a full range or schedule of what we try to accomplish in that year, which developers are working on any number of features around those times, so within there we try to break it down into ‘Ok, so this period of time we’re [going to] concentrate on area X and do the features there. Or there can even be things like performance, so in this block of time within that full development cycles, we’re just [going to] have an X number of developers doing performance and getting that done. So it’s [kind of] of a fluid organization to the whole cycle but we try to keep it somewhat managed to compartmentalizing what we are doing.

What do you mean by managed?

Just manageable in the sense of when we start a whole release its just not undoable, un-testable code all the way up until end date. We try to section it off so we can within the full development lifecycle, get to these certain points where ‘Okay, this is something that’s complete, is testable. The code isn’t totally having tons of loose ends all over the place.” Is a manageable little chunk that we can get to and say ‘ok, we got this far, let’s go on to the next step.’

How long does an iteration last and when do you actually deliver the product to the customer?

Yeah, that’s probably where we actually deviate from the Agile. The iteration we complete has a testable product but not one that we would release to customers. It’s a very much still in alpha phase and features may be modifying from there, they may go on to usability test and if they find things out from user’s bed we’ll need to go back and edit which will affect the test plan on that feature. So the only real customer facing release is when we get to that final goal of the entire cycle and all the smaller iterations within it are more for making it testable internally for QA, for usability testing, for internal beta testers and things of that nature. The way we facilitate that too is the end of our last release, which is the product that goes to our last release – that is actually branched off from the code base so that’s its own snapshot of the code at that time, while development for this new cycle continues on in the code branch. And what we do is, if we do need to release something to the customers within that cycle we need to make the change in where we are developing now, integrate that to the branch that we released and do another customer release from there.

How do you track the timeline such that it is finished within the scheduled time? What do you do if the schedule is not met?

We have a product manager that essentially keeps track of all the timelines and what we are estimating. The weekly development meetings, that's one of the Agenda items is to see how we are progressing, get what we estimated for this development and see how close we actually are. In the iterations too, we have these targets of, well, we like this iteration to end at this period and see these things finished by that period. It's all very speculative. All of it was estimates for time periods, all dates drawn the engineering estimates. So, if your estimate is off that you estimated at the beginning of the period, it takes you longer. All of our dates are shipped accordingly. Its all driven by the features, so we don't say 'By this date we need to see business done and if it isn't that's the end of it' Its more 'what is feature going to take …. Well, I think it will take this much' and if get to that point and we figure out 'Oh….well its [going to] take more because we didn't know how to fix xyz.' It sets off from there and applies to the overall date to where we say 'now we are looking at an additional X number of weeks on the end as well.'

Can you describe the workflow for each release/iteration you define?

Yeah, I explain as far as I know because I don't actually have much interaction with that side of it.

We have a product manager, plus we also have a project manager. And the product managers are the ones that are gathering those requirements for features or development or modifications to the existing features all from feedback from users, tradeshows, going and getting talks about the product. You know anything out in the user community they are gathering all of those items. They gather those up for when the next cycle is [going to] start. At that point the product tech lead goes through all those hundreds of ideas and features and tries to categorize them into the main development items that would satisfy a lot of those at once so a lot those same feature requests can be satisfied by one new feature or one change to the way the application works and we break that down to see what common traits and what major areas we have. From there the development team gets together and starts estimating those large chunks of features, and that's done in a group collaboration so people can weigh in on what they think if that's a wrong estimate of if they may know of some technology that may implement faster. That's all weighed-in in a group process, so once we have estimations on all of those new areas, then it's a matter of with the project manager coming up with a timeline of … you know.. if get all of these back within, we estimate to release 3 years from now or something ridiculous, you know they come up with the overall time estimate and say 'that's not very feasible and people or these user would like to see it in this time frame or the sales opinion would like to release something for this tradeshow at this time.' He comes in with all the time constraint factors that might be able to pin point what's the best; that is the project manager.

The project manager deals with mostly the timelines and things like that and product manager deals with gathering all those feature ideas. So we having estimated, the project manager comes in and says here would be a good time to release, or here would be good time to release based on all these needs in the community. So based on that we break down well if we don't have this much time then features xyz fit into that scope because are estimated to be this amount of time. So once we start getting, flushing out an idea of bringing those together, the estimates of a likely release times of the needs of the users. It all kind of boils down into this timeline of what we're [going to] work on, what we're trying to finish by. So from then, either at that point, or when we are estimating, design docs is written. I'm actually not sure specifically on that process but within there, at some point design docs are being written either to help estimate or flesh out what's actually going to be done.

And then once we are through with that process, we'll begin - people either pick out these areas that they [want to] work on or they have such expertise in that area that it's natural for them to be working on whatever task has come up with. And development starts, I guess also at that point we break down the pseudo iterations, where we say 'ok, all these things we're working on kind of are estimated to take this amount so lets see where we are at end of here and that we'll call the end of the iteration' and say 'ok, what's it looks like at the end of this iteration at that point are we completed, is there something we could get to QA, how's our schedule looking overall. So that continues on throughout the cycle of I'm doing the work, getting to those points, re-evaluating, continuing on going all the way to the originally estimated timeframe.

So where do the QA start playing a part?

It is actually all the time. By that I mean, we try to get through these milestones or iterations where there is a feature to test but leading up to that point is where we'll either be asked by development to 'I'm working on this, so I could really use this utility or something that automates this process so we'll work on that within that period. If there's not anything specifically on the product to test, then we can look at what things are going to be going into this product and start to build up test harness or automated test framework that will, when you get to that point, when it is ready to test we can start writing these automated tests on it. So its really continual process with various types of jobs not so much just directly testing the product either improving our QA process with automated tools or helping the developer directly or getting out the product and doing actual hands on testing of the product.

In this workflow, do you use tools to enhance the process? Where? What are they? What do you use them for? Why do you need them? How often are they used? Who uses them? Have you ever run into problems using them?

Yes, certainly. We have source control obviously, and lots of tools built around source control that help with peer code reviews. So before we check in we have all these systems that allow the code you [want to] check in to be reviewed by another developer and just as a double to make sure what you are checking in is readable code, is good programming style, is there any obvious flaws that you might have overlooked, things like that. Certainly on QA side, we have defect tracking, a db to store all that, as well as some metrics pulled out. As far as open issues, new issues opened, fix rate of defects, things like that. We certainly use, profiling tools, things like that, that's more at the developer level not at the team level. Those are the major ones that we deal on a day to day basis.

Can you explain more about the automated code review process?

It's generally a system that is tied into our source control so it can detect check-in, by detecting that check-in, it can notify another developer about the code that's being checked in. In which case they can look through it, and then we also have tracking systems where they can actually validate that that 'I did look at this code' so it looked ok to them. There's also checks on you're making sure that you get the review notification from that other developer before you check-in otherwise there's that nag email that goes down, so you need to get your code-reviewed. So it's a four track process that knows when you check-in code and knows that someone other than you actually looked at the code.

Both the Source Control and Defect tracking system are pretty standard as far as most development tools use.

How does your defect management process work?

The workflow is technically logging a new issue which will automatically be assigned to a developer, that developer can re-assign it somebody else if they want. Typically it's the correct developer, so they will go through and make the code change to fix it. That defect report will be marked as fixed and actually keeps track of two statuses, the status of going from assigned to fixed. And while it's fixed, it's also categorized as ready to be verified. From that is either goes to the status of verified that the bug was fixed or it can be re-opened and the other status will go back to assigned. So that's how QA gets the notification, seeing ones that are marked as fixed and ready to be verified.

I should also mention the automation tools that we use. We actually have a bunch of tools that have, they are separate and we use them as a validation of automation. So we don't just have one tool that does all the automation. We have to actually tailor specific tools to do specific things on the product that we [want to] do and that involves some there's test tools that we have that concentrate on the 3D graphics and rendering fidelity in the application. That runs by itself, runs all those checks and get a validation score out of that of that particular set of the application is working. We also have a scripting engine within our product so we have a testing harness that just explicitly tests that. So it's a bunch scripted tests that run within the product and to make sure that the scripting engine is working correctly. And then we have tool that just specifically tests the graphical interface of the application, so that just focuses mainly on if the input boxes are correct and if the buttons are doing what they are supposed to be doing and dialogs, things like that. And also internationalization of my category too, so we can go through the application in application in any language and make sure that the correct internationalized strings are within the interface. Make sure nothing got missed as far as compiling it for internationalization, things like that.

So separately they all do their own things and then we realize you know collectively out of all those results what is the quality level of the product.

Are these tools developed in-house or off-shelf?

They are mostly home grown, so we build them to our needs. The actual UI tool is off the shelf or semi-off-the-shelf, where we are using the off-the-shelf software and we have written our framework around it to actually drive that off-the-shelf tool. But at least for us, that's the only 3rd party tool that we use. Everything else is all custom made.

Why were they 'home grown'?

For our specific case it comes down to not being able to find the right tool that would do, being a heavily graphics application there is actually not a lot that facilitates automated testing in that realm. So a lot of the times we'll in order to test that we'll have to come up with tools that talk directly to the graphic APIs or the operating system APIs, to get something that we can count as a test tool.

Do you communicate with other teams?

Yeah, so within the company there are certainly other projects going on which is a help. I talk directly with other project teams in the area of automation seeing what tools they use or what ways they have come up with as far as automation to test their project, and just get out things we might be able to do or other tools out there, other techniques things like that.

How do you facilitate these?

With video conference it can be face-to-face which is nice. Other times it will be through email or through instant message.

How do you track your requirements as you are getting them done?

Ah yes, that's good question. And I'm not certain I have the answer for ours, because that kind of lives off somewhere in its own world once the design docs are written based on the requirements are originally come up with. Those live somewhere and I’m not directly, I don’t directly interface with that a lot unless I’m testing something specifically not sure if it was supposed to do something or not. I’ll go back to the developer and if he doesn’t know then we go back to the requirements doc. I’m not saying that’s a good way either, its something we’d like to improve upon is making those more visible to everybody and adhering to those more and making those very comprehensive. But we found that a lot of times, we’ll think of these features, or think of what we need for development, or write these requirements and once it actually gets implemented in the code, its not something that we figured it will be or when we take it out for usability studies, its not at all useful the way we thought it will be so they, the requirements, are typically get written at the beginning as the idea of what we are doing things and just typically don’t get updated along the way. We realize something we’ll have to implement a different way we’ll just do that and not actually put back in requirements. So that’s certainly one area I think most of the people around here would like to improve upon it just hasn’t been a priority. It hasn’t received any problems because we’re very used to it, to try to make it into a more important process.

What changes did you make to the Waterfall process to tailor it to Agile techniques?

The only real modification is just being able to categorize it into these pseudo iterations, not really a strict iteration but more using the idea of an iteration to group chunks of work together, and to have an end date where we can look at it and say how the development is going at this point, what it looks like at this point. That’s probably the only real feature of Agile that we incorporated into ours. As far as I think, that’s just pretty much it, and that’s just mainly for managerial organization and keeping tabs on what’s going on.

Do you use object-oriented design? UML?

Yeah, our code is C++ so we rely heavily on the Object Oriented design of that language. We don’t use UML. I’m not sure why not, nobody is had an interest in it, to actually try and use it very effectively.

How do you capture these design ideas?

Mainly just in text-based human readable descriptions of what we are trying to do, diagramed out. If we do diagram something out, it will be PowerPoint or office graphics. That [sort of] thing. But a lot of times it will just be explicitly written out in English grammar of what we’re trying to do and how its [going to] design and some of it comes from non-programmers actually weighing in on the design reviews in systems where they are our UI designer will say in the design doc, this button should do this and we expect the result will be this. So that gets incorporated into the design doc and the engineer will say ‘well in order to implement this we’ll need to create class Whatever that interfaces with class Y and such.’

What languages do you use for development? Do you use an IDE for assisting in the development? How effective do you think it is?

C++, yes, we use an IDE, I don't know I can really say what they are but its one the most commonly used on Windows, I would guess. And we do cross platform development so we have the common IDEs on Windows and Mac.

How do you manage traceability between requirements to design and design to test? Do you use tools to manage this? If yes, what tools do you use? If no, why?

It's not rigorously traced. So we'll have requirements and then design docs and that's kind of the ideal that we're shooting for, so that will progress on and the developer will get into development. Oh this is much easier way to do it, oh I didn't realize about this, I could've done it this way. So we deviate off from those requirements and design doc ideals. When implementation is done, when it gets to QA we'll actually talk with the developer first as far as how it is supposed to work, knowing that it could've gone through modifications like that; more so than going back to the requirements. I think a lot of that has come out from experience where we have seen something that we don't think was specified and go back to the requirements and we say, this is not how its supposed to be in the requirements. Then the developer will say 'well, that's because I came across this and I did not take the requirements, this design was what I was looking for' and he went to talk with the UI designer or the product manager and they said 'oh yeah, maybe we should do it this way.' So it certainly deviates a lot from requirements along the way and that I think its just being the nature of heavily graphical application; to describe that at front will be really difficult to do in order to foresee all the things in the end in the way this product is going to be implemented. You can't really understand how well it will interact with each other, so its actually implemented and see if this isn't what anybody wants we could all copy it in the requirements, but….

Do you have any continuous integration set in place?

Yeah, I guess that's another tool I should have mentioned. We have a continual build system that will kick off with essentially every time a change goes, but if the change goes within the time its building the next will build with all those next build. So today, each check in will get incremental builds and we do a nightly build during the evening, so that we can see that nothing broke.

How often do you run your tests?

We have test verification the runs with every build, and that's very minimal so that will just test basics of installing the product, launching it, building the document, shutting it down – all the basics. So that's the only one that gets run repeatedly, as well as actually, we just started implementing unit tests, which is reverse way we have several ways of good looking code written and we're just now have started writing unit tests. But we do have small amount of those that gets run with every build.

How often are the automated test framework run?

For those we will run typically once the development gets stable – at point where the developer says its ready to test – then we start running those. Its more of a manual process of kicking it off and it will run through all of those tests and we will see the result of the tests in the end. So those don't have a defined schedule and those are just more run as we get to the end and we have a release candidate, then we'll run everything on that release candidate to make sure everything still passes – all the tests that we have with every system.

Are there any external entities that influence your process – e.g. FDA? How do they influence the project? How is this influence handled? Do they affect the design process heavily? How about the testing?

Just I would say sales and their influence will come in on you these ideal release dates for when we would like to see the next release happen. Certainly, users are influences if they find terrible bugs or some form of bug in the product that's typically some of our expert users will find this block in their development that they work for a large company, then we take developers off the current development and fix that issue to release a patch for the current product. That's all, these are the ones that are noticeable and that influences.

How formal or informal do you think your development process is? Why? Can it be improved? How so?

It's really informal, I would say. It really kind of is fluid as far as reaching the dates we [want to] reach, or like I was mentioning implementing features off the requirements. It really is flexible and doesn't really stringently apply or requires by any strict structure. Which is good and bad, but in a good way it certainly helps the nature of our product being a graphical application that artists use and artists have specific requirements and how they expect software to perform and be used so I think our development process somewhat reflects that and we do need to be less rigid in the development process in order to get a product that can actually meet the standards our users require. So the bad side about it is that is then every time things deviate from the requirements or from the process, that's just higher risk of bugs being introduced. Security issues, certainly communications and things like that that.

It can always be improved. Big question is where you should concentrate on improving it. And the biggest area at least that I have identified is automated testing. Before I started working on this product all of the testing was actually manual and hands on testing. And that certainly allowed bugs to get through just because it was not scalable enough to find everything that could possibly go wrong if you do have flexible development process. So at least for me one of the areas that I thought of improving was introducing automated testing that can give us that coverage and if the development process does deviate from requirements we can then have these systems that can run hundreds of tests and catch those kinds of things.

C.6 Interview F

Starting off… can you tell me more about what you do? Your problem domain? What issues are you trying to address? Do you directly interact with your customer?

I'm the team lead for the development of Company F's file system software. The software is a next generation file system, so my role is primarily one of a combination of development - I do a lot of development as well as coordination of other members in the team. The team size around 15, that's just the development. Actually it includes the management infrastructure, there is some small form of management infrastructure, there is a large testing team which was not included. So yeah, we include all of them.

How big is your team? What are the different roles on your team, e.g. Testers, Developers etc.? What kind of interactions do they have and how do they have it?

Within this company, it's a pretty informal team structure, typically. We have some set of developers on a given project and they typically will have some assigned roles or areas, in particular in a large project like this one, its sort of specialized in some parts of the code. And then other team members who may have more generic knowledge, and then of course you have infrastructure support like management is overseeing team and possibly programming management's responsibility is to keep program on track for schedules and then for larger project's we'll have testing engineers who are responsible of testing a product.

So again, in my role as the lead I am interacting with most of the team on a relatively regular basis. We have regular meetings every once a week. Although as a geographically distributed team those meetings are not face to face all the time. So usually we try get them at least via conference in the team meeting, although sometimes its just a phone conference. We try once or twice a year to get the entire team together in person, try to get them in one location over for a week together. And so, and then of course there is subset of team which is here on this floor. Those team members I interact with on a pretty much daily basis discussing various issues and that sort of thing. And then I also get them interacting with other team members who are not geographically located here on a day to day basis. The test team a little less interaction on the location directly. Most of our interactions are through email and then we have our features set up for getting testing done. For submitting test requests and getting test results back, usually how the interaction goes. But occasionally you have the direct interactions, it's a little more awkward there because our test team is located in Ireland and in China. And so the direct interactions are awkward at best of time.

What development life cycles do you employ? Why do you use this lifecycle for your team? Does your team size or structure or project needs influence the lifecycle?

Our sort of development life cycle process, well, this company as a whole, so… this file system is part of the operating system of the company. So when we deliver the file system, its part of that bigger thing. The operating system as a whole is has a sort of train life cycle where we have built, which occur every two weeks. And when you want to add something, add code to the operating system, you integrate into a build and then at the end of the two weeks period, you add to the code and in that window of two weeks we'll build it and that represents a relatively stable point in the operating system. Its thoroughly tested. If there are problems the code maybe kicked back out, but then you can put something new in the next window, etc until you have these two week windows where code is going back in and every two weeks a snapshot of the build is taken and thoroughly tested. Now that's sort of the general model of development for the operating system. For a given project now, when we first started developing the file system we were not in the operating system yet. We were an independent project and so as we developed the file system we actually adopted a very similar model for our own development. We sort of had a what we called a gate. The gate is the stable code base. Individual developers will take children of that gate, develop in that context, and then when they had tested their changes, their delta, they will put back into the gate and then periodically we would, the test team would take snapshot of the gate and run a battery of tests on it. And so there's a funnel of changes coming in through a central gate and that gate is tested periodically. Or build in windows by the test team. It's a little bit chaotic because we didn't have a fixed schedule. Now we have this 2 week time train, it was more like 'well things look stable, it looks right, lets customize'. Once though we have put back into the OS, we didn't put our project back into the kernel until in general – when you generalize the model in this company when you're doing kernel development the expectation is that the code that goes into the gate of the main project is always stable. Stable as possible, stable as you can make it. And so we, in theory, would never put anything back into the OS until we're satisfied, the code is in pretty good shape. And that was largely the case when we finally 2 years ago put the file system back into the OS. The company is kind of unique in that, roughly a week after things go back into the OS they become visible on our product website. So it becomes publicly visible very quickly. That's one of the reasons we try to make things as stable as we can before we put back in the gate because it becomes visible to the public once in. So the idea is you test your changes thoroughly before you put it back. So testing is very important. The file system is pretty self contained so we can test it pretty thoroughly and put it back. On the other hand there are things you cannot test completely because you need the context of the entire operating system. That's why you have to put things back and you have these two week gate windows where you test the whole thing. That's where you catch things in a larger context, in a larger scale. Yeah, well its good inside your code here, but we run it inside the zone, or in context with another piece of code over here then things broke down. So that gives you levels of testing to go on. So once we put back in the gate though our development on it continues because products like the file system and a lot of other enterprise is continuing development life cycle. I mean just because we have something stable that was useful and we can put back into the OS doesn't mean that you are done. There will still be new enhancements made to it, and of course there are bugs discovered in it overtime, those will be fixed. So once its back in the OS now, its been in there for 2 years, I'll still periodically take a bug or a enhancement and work on those changes, you know develop my own workspace, test it and put it back in the gate. And that, the file system would do it, individual developers do their own put backs. Some teams within the company they all coordinate a set of changes, like the team will sort of work on a bunch of stuff and they will centralize it and coordinate it into a single bundle that will be put back in. In this team, we're more independent, we pretty much say 'you've a bug. Fix it. Test it. Put it back yourself.' So we don't do these massive coordinated put backs in the code. The put backs you see going back in the code base are on a regular basis by various team members. So it takes a few days, and some times one week.

Are the put backs done in parallel?

So its semi chaotic. The code management in within the software, the way it works is that, you have the stable base, the gate as we call it. When you want to do a put back its your responsibility to make sure that your delta change is coordinated with the delta level you're putting back to. So its all SCCS based. So you should make sure you know, if you're putting back version 10 then version 9 is the last one is the gate, that somebody else hasn't already put 1- in there. So that's one thing you check and when you are ready to put back, the gate is locked down momentarily. You do you delta put back and the gate is unlocked. The gate is locked down for a few minutes while you're doing your put back but that's it. So in general its pretty straight forward, its pretty easy. You're, you know, if you're putting back something that's in your code and you're certainly coordinating with the other team members, and somebody else is working on the same file, you're saying, I'm doing my put back now and you have to merge after my changes or you put back, that sort of thing. So its relatively informal. For example, if we put back the file system as a whole, it represented a large delta change to the OS. When we did that we actually had what's called the gatekeepers, who…they lock down the gate. They say, 'OK, there will be no code changes for' and we had a basic lockdown for a weekend. And we were then able to make sure that everything was merged in correctly and we were able to do our massive put back without any disturbance. Because what could happen, I'm now ready to do my put back. I've made sure everything's okay. And then somebody puts in their put back and I'm out of sync again. If you [want to] make sure that everything is correct and the more things you're putting back, the more difficult it is to make sure you're merging in. So you know, its partially a process of engineering communication and partially a process of the software itself, maintain locks etc. So you don't can't to rely completely on the software a lot because we don't [want to] have to lock down, we don't want any of those engineers to lock things down for long periods of time and forget. They could lock the gate down for a day or two and that would be like serious problem because there's a thousand in here working on this project so they all want to be able to do their work. There is like an essential authority like the gate keepers who are managing things and that's not a team of people, so it's a small team of people, like about 5 people are trying to make sure that everything coming back is in good shape and that itself is great, they perform sanity checking, every time a change comes in they do a sanity check on it to make sure everything looks okay. I can explain a lot about the whole process of putting back and how stability is maintained. The gate software is as I said managed by this small team of about 5 people which, so the gatekeepers are basically people who are managing the stability of the gate itself when its put back to make sure that no one makes a mistake. They are trying to make sure that there are no mistakes made in the put backs. Then there is a technical lead though who is responsible for making sure that the content that's going back makes sense. So he is mostly hands off and just paying attention to everything that's happening and raising a flag when he sees something funky happening and like that 'that code shouldn't be going back'. But its really very rare that he has to step in and do anything. So that's at the operating systems level. Within our group we [sort of] have a mini version of that. When you want to put something back into the gate you need to call the request into the gate to give us permission to put it back and there are small number of the team who have the authority to allow a put back into the gate. So although I said, you know, any member of our team can put back, they can't put back until they get permission to put back. So what happens is some team member says 'alright, I've fixed this bug', its been reviewed, I got someone to look at the changes and they tell they have to let the others review the changes and sometimes they even want more that one person to review the changes. Then they have to request to our small group that alright, I'm ready to put this back into the gate. Then we, that is a me and a couple of other team members will get that request, look at the changes and say 'yes, that's okay. It looks okay.' And then you grant them the authority, the privilege to go and actually do a put back into the gate. So there are several steps you have to do to actually get to a point where you can put back. Its not just within our group, within the project, on top of that there is the gatekeepers who are in charge will also be looking at it. The tech lead of the whole product will take a look at it. But there's so much stuff happening on a given day. On a given day there's [going to] be thousand lines put back into the code base. So the tech lead cannot possibly be knowledgeable about everything that happens but….

Are these permissions granted electronically?

Yeah, so although I described it as sort of a personal 'Yes, you can do this' its all managed through software. So as a software system when you are ready to go and do something you submit a form, rum the RTI tool. And the RTI tool sends me an email that says someone has requested to do this and I'm on the tool and click to say 'Yes, I can do this'. So its all software driven…

Is this RTI tool home grown on third party?

I believe that it probably is. I'm not certain of that. Much of some software, code management software has been home grown although we're beginning to migrate to some other systems which are not home grown. Which are produced by a company.

What we have now, the software management system we use now… can't remember the name of it. So we have some home grown software management infrastructure tools that we use for the primary work now. As I said, that's about to change now to a new system. And that's actually been a relatively difficult process. A new system was proposed over two years ago and it has taken at least as long just to get things migrated over. So there are casual tools for put backs. There's the software management system itself which handles the put back itself and merging the tools together and that's built on top of things like that CCS which does the actual code management. There's also a whole bug tracking system which we use called BugTrack for bug management and it ties it all together. So the way things tend to work is that once we put things back into OS then someone is using that software so various people within the company itself, got engineers who are running the software on their desktops but also, as the release [is delivered] very quickly and so gets used in the community very quickly. If someone out there finds a bug, then they submit a bug report in the BugTrack and eventually it'll pop up in my email saying help us reported bug found in your software. So its categorized with what part of the OS – kernel, file systems – is effected. And that, sometimes it takes a while to get to me because maybe someone who doesn't know what happened to the file puts into a more generic category under the OS, until its looked at in the kernel and decided 'oh yeah, this is in the file system. There's so little time to look into it, so he'll just push the buttons. The ones coming to me or one of my team members get far and out, so a couple it gets to says, okay 'this is in your area, you should watch some of this stuff.' And they are categorized in terms of severity and priority. And we do it ourselves, although committers will be giving some guess and we'll end up re-categorizing it. And then I determine which bugs to work on. And then the same tool handles bugs, its called RE or Request for Enhancement which are for new feature request. Any given moment, we definitely have now several hundred bugs in our queue. And a project like this file system that's not unusual. The project currently has grown to about 70,000 LOC. So, it's a lot of code, lot of bugs, lot of feature requests. Although this build required developing very little bugs. When you have that much code there's problems in several different places.

What do you mean this process is like a train?

So the train process is, for a said bunch of software, is how you add code into the OS. When you talk about the train what I mean is that, when I partition into these 2 week periods where at the end of every 2 weeks we do a build and then test that build. And so what I mean by a train there is that you're sort of imagining the product speed past you in terms of build, you say build 1 and then two weeks later build 2, build 3, build 4 extra. So for example, we're currently on build 80 of the new version of the OS which is not yet productized. So the term we use in this company is that a train is going by. The number of cars is the number of builds in the train and what you do when you are developing a product for the OS is that you target a train car. So for example when we get the file system, we targeted build 27. And that's the car we went into in, you know, the OS train. So we got in at build 27 of the OS. Other projects will target other windows and its important to us because what happened is our designated cars, train cars if you will have special significance. So for example, when or if we ever decide the OS is [going to] finish up, there'll be a point in which we'll say 'OK, boom no more big projects can come in anymore', if you wanted to get wise. So basically they'll let you know quite a bit in advance but certainly in the OS its like 'Oh, gee, we've to get in before build 1.3 or whatever'. So don't need to worry about that. Also there are intermediate points which we are going to produce what we call these express releases to last, they go out in the OS which are supposed to be… So along the way we have intermediate points which we want to have increased build in the product and so again at those points, there will be a window of builds where we won't allow as much change in. Also we may not, we'll require for example, more careful, more thorough evaluation of the code, so as I said earlier, that at [27] you put your code change back so have to get someone to review your code change. Then at some of these special train cars, you may have to get two people to get two people to look at your code change before you can put back and that sort of thing. Also you may say well only P1 bugs, priority one bugs, will be in this particular build or only things like that. That's speaking that we're trying to focus on some particular aspect and increased stability and tackle the most important bugs and that sort of thing.

So who decide which features go in which build?

That is largely determined by management, or a schedule that has been established for things. So I said when we produce on a regular basis an OS express release, which is considered to be stable release of some of our OS community releases and so these have been sort of pre-scheduled by a combination of management and marketing and whoever gets involved in things like that and someone makes a decision roughly every quarter and so those have been scheduled. So then as a result they work with the gatekeepers for 'well, we want this to happen', 'OK fine, we'll make sure that this gets in the train during these builds. So it depends where the specials come from. Usually it's the company decision at some level where they [want to] release something special and more stable.

Can you describe the workflow for each release/iteration you define?

So, lets pull back – we're talking of several different levels here. So there is the massive thing which is the OS, there's the smaller thing which is the project and so I have put my own schedule and pressures etc with respect to just the file system in terms of what I'm trying to do, what I can accomplish. So for example, a month or so ago there was a big push to get a set of features in the file system put back in the OS. And so, and actually that process started months before that, maybe six to eight months ago, I was asked to develop about five different features for the file system. And at that time, actually those requests came from a combination of management, actually more directly, more accurately, I'm in management. Those requests came from other groups within the company who wanted to leverage the file system for their own products, but they needed additional functionality in the file system to be able to that. So they made their requests to us, but made it towards the management.

Management is their communication channel if you will, although that's not entirely accurate because reality is that we talk directly most of the time but it officially goes through management channels because its all about allocating resources in the company and that sort of thing. So they made a request to us saying, 'could you develop these five new features?' We said yes and we said 'When you need them?' 'We'd like them by this time.' 'That's great, but when do you actually need it?' 'We have to have it by this time.' And we have a developed schedule which would target that window of when you would like it and when do you have to have it. And then we work with our management to sort of say 'This is how much resource its [going to] take, how much time its [going to] take'. So we work out some of the schedule, although in reality that's what management would like it to be fully scheduled where we say exactly when we'll be doing what, what's [going to] come in then. The reality of software development is that you can't possibly do that, particularly future development. So we say 'well, I am pretty sure I can get this feature developed, this part of feature developed in this time frame and he's [going to] develop that part of the feature in this time frame.' And we make sure we can get all of this done by this week and by the end of August we'll pretty much be there. And then when August comes around, we realize we're nowhere near where I should be done, so management gets all upset and we sort of say, 'well ok, give me a little more time' so they drop date is not until October so we have more time and then we do a lot of work coming, work late hours to get everything done and so. And that's the reality of how software is typically developed. We try to plan out exactly what its [going to] take and we try to estimate accurately how long things are [going to] take and keep to some sort of schedule but for the most part it's a chaos that says, we really need to have a feature put in by this time and you've got to test your bugs and put it in, and sometimes its like 'Oh, no problem. We can do it' and sometimes its like we can barely make it or sometimes … And that's what the latest projects are like you know, companies promise functionality and actually don't deliver it a year after that because you really can't be complete until then. So we were doing our development of these features and it's a lot of combinations of sort of working out in the code what needs to be done, working with team members to try to get things done. Because you know for large changes, large delta changes are complex and some people are working on part of it, some people are working on other part of it. My role is often just sort of make sure that everything is fitting together right. In particular, the feature being delivered is right for the code, we're very particular about in this product that the code remains clean and stable. So we spend a lot of time, reviewing code changes and trying to make them as good as possible. They stop when there is huge pressure to produce a large change very quickly and stuff creeps into the code base, which can't be avoided. But that's true that again in every software development and projects I'm aware of. And what's a little bit unique in this company and it's the way we work is, almost always its largely engineering driven. Management says its role is to be mostly agitated about these things. Try to control us and offers prizes or bonuses and stuff to be delivered and that sort of thing. But when we were doing these features I knew exactly which engineering team wanted them and I was communicating with them on a regular basis, what they really need delivered and when they need it delivered and that sort of thing. And that's really important in this company is this – a lot of very direct communication between lots of engineering groups who are actually doing the development. When that doesn't happen you tend to get a breakdown and you don't get what you want. When its too much management in particular, you tend to loose the telephone right, everything gets garbled and you don't what they really want so you have to go talk to them. So this company is a very engineering driven company.

In this workflow, do you use tools to enhance the process? Where? What are they? What do you use them for? Why do you need them? How often are they used? Who uses them? Have you ever run into problems using them?

So a variety. Out test team is critical to us. We have a group of about five or six test engineers and they have a battery of tests, so they have a large number of test programs and test harness they developed. This file system is now been in development for about 6 years, 7 years. It evolved quite a bit and so our test team, we actually had a test team with our project development for quite a long time, holding on for the first few years. We got a test team allocated to us and these guys are really good, they've been developing a lot of big test harness to get around the software. So they run battery of tests and what happens is for example I'm [going to] make some change, even if I'm making a relatively small bug fix, I think it'll have some ramification somewhere, I'll send it to the test group and say 'Hey I have this – my new version of code, could you test it for me?' and they'll have it for several days and run a large battery of tests. And so they have a large number of machines that they run their tests on and they are typically running their tests for various engineers simultaneously, so if they are running 4 or 5 tests at any given time for various engineers. We do a lot of individual testing on the bugs. Its good to do it that way because you know when if a problem shows up its [going to] isolate to your fixes as opposed to another model which the other groups often use to combine a bunch of post bug fixes and send of for test, and when something breaks what we tend to do is our own individual versions and the test guys are kept busy because they are running more tests at once but when you just get a bug back its like, 'okay you should test like this, because you only had my changes in it.' So that's critical to testing. We also utilize some very useful tools that have been built into the OS itself, I mean what we're doing in the file system is kernel development. So we're working inside the kernel. That means that when it breaks, it crashed the machine typically. Major tool used is B-Trace which, I don't know if you are familiar with B-Trace, is basically is a tool for monitoring arbitrary things in the kernel or not even the kernel but any process. So B-Trace is a very useful tool for us, for this. Finding out what's happening inside the code and then the other major important tool is called MDB, module debugger, which is the tool critical for analysis of a fixed state when you are trying to figure out why the kernel crashed down and it can be used to look at it and figure out what's happened. Most of our bugs that we get, we get bug reports back is someone, well I was doing something with and it crashed and so they give us the core and we use MDB to analyze the core and figure out what happened. That is critical to us, because typically we can't reproduce whatever they did to make it happen. Either it was too complicated or they had some particular hardware set up that we don't necessarily have our hands on immediately or you provide the code running inside their code or whatever. Often we can't reproduce it directly, so instead we have to take just the core and figure out what happened and make the fix. And our test guys, like I said, we don't typically run the tests, I run the test harness myself and so I don't have to take the whole to understand the core they produced, usually I can figure out what happens from the code and then we test it and so it makes my job a lot easier to do it that way, so. Post mortem analysis is critical in kernel development. But when that's impossible, that's when B-Trace comes in and that's when you can't go post mortem, [mutilate] it and B-Trace is really programmed to figure out what's going on. On top of that we also have user land versions of most of the code. So the product is a file system but we have, we build it in two forms, we build the internal module and then we build a library module for it as well. And then the library module is run under special test code directly. So we don't have to run in kernel context. And so we have a program called V-tasks which goes on top of the library module which just hammers on code basically trying to call the interface, trying to set up all kinds of weird things and stress it as much as possible. And finds out all our problems as well So we build and sort of run it in user land and crash it and do kinds of things to it without having to worry about the hardware, machine crashing and that kind of stuff. Its critical to us as well. It made getting some major bugs out of it much-much easier. Architecturally the file system has been posed as sort of three nature components, 2 of which are very complicated. One of the two you can actually test outside the kernel, and that has been a very critical factor in getting things stable.

How do you manage traceability between requirements to design and design to test? Do you use tools to manage this? If yes, what tools do you use? If no, why?

So when a new feature is requested, usually the requirements will be listed in the feature requests. So someone makes feature request saying 'I want this new functionality added in the file system'. And they'll file a bug report in the RV actually. And they'll state 'here is how I want to work it.' The RV's are the requirements in the train of how its supposed to work. Now that doesn't mean necessarily, there may be give and take 'cause now what happens is, the senior engineers of the project sit down, if it is a major new feature. We'll sit down and say 'Ok, you specified you wanted this new feature.' Sometimes its like 'yeah, that's good. Alright, we understand. We'll give you that feature later.' Sometimes its 'well you've asked for something that doesn't really match architecturally what we can provide to you and we'll have a give and take about what you really want to do. What is the functionality that you're really looking for here.' So we'll try to refine the requirements to be more realistic or more accurate to what we can do in the file system. Often when somebody requests something of us they have a very specific model in their mind about what they are trying to do which is often based on another file system, some other environment and that may not match our architecture and we'll have to rethink a little bit and say 'Well, you're asking to do this but we can achieve the same thing by doing this. Will this be sufficient and then yes or no. If no, okay, how about if we do this' and again, work out some compromise. So we'll work with the requester to derive the real requirements. Once we have the requirements we'll captured inside this bug report and the bug will be modified to testable requirements. Then the actual work is going to be performed by some set of engineers now. Typically at the same time the test group will be informed that this is the new feature functionality being developed and they will start developing tests for those new features. And so if it's a new interface with new functionality they'll start developing test sub suite for that new functionality. So while we're off developing the actual functionality, they are off developing the test suite for it. And then when we try to put back we [send them] the code and they run the test suite on it to determine the functionality is actually being delivered. There's always room for misunderstanding about in either party about what is going on, I mean 'actually what you're testing isn't what we are providing' and then we have to go back to the actual submitter and say 'for verification on something' and ultimately we may even get to the point where we put it back and someone will say 'well, that's not exactly what we want.' In some ways the requestor's functionality will be the ultimate arbitrator there. But sometimes the case even what they request and what we provide may not match exactly but it's the best we could, or is in our minds the better way to do it. So, we as the engineering team take certain liberty there where we can determine sort of ultimately what the functionality included is all about. And yeah, there's certainly room for some sort of misunderstanding somewhere along the line but that really doesn't happen very much.

What languages do you use for development? Do you use an IDE for assisting in the development? How effective do you think it is?

C. Everything pretty much, the OS's kernel is almost entirely written in C. Some small amount of assembly, some small of C++ but, its almost 99% C.

As for IDE, no. Not really. As I said, there are a set of tools that are used that are loosely coupled, there's nothing that's really significantly integrated. And I think to a large extent its mostly a matter of practice in terms of what you do and what process you go through and that kind of stuff. There's a lot of tools that can pull aspects of it but there's nothing which sort broad over reaching tool which makes sure that all the things have been properly done, because it's a little bit too undefined. Within this company we have this sort of over arching tool which can actually control it completely. Its one reason why we have to have a lot of human involved in a lot of places to make sure things have been done right. So it will be nice to have sort of a complete development environment but we really have a complete development environment but its just with a large set of tools that you can wheel and you need to know how to wheel them and what the tools are. There's nothing that's guiding you really directly. There are humans that guide you in various aspects. There are project leads that are showing junior engineers how to do this and what is the process and how do things work. But one of the early things you learn as Company F's engineer is what is the train, how does it work, how do you put code back into it? What other tools are there, what are the commands I [want to] run, when do I run them and there's a lot of sort of informal process. It has been formalized but in much ways its relatively informal and so its really then [sort of] evolved over the years by engineering groups over here at the company. [Changed] by the teams and revision engineers, and as the tools come along they get integrated in and the old tools get [sort of] of shuffled out. But it isn't controlled, it isn't integrated.

Earlier you mentioned that you were moving to a new software management system, can you explain what it does?

[Yeah, its called] Mercurial. So both Mercurial and the old one whose name is FileBench. [I think.] They both are sort of software management infrastructure tools. They are intended to help you manage large software projects. So, their primary use is for sort of managing large versions of things. So the operating system is thousand and thousands of lines of code organized in a hierarchy and I work in a particular part of the hierarchy technically, you know, but its quite a complicated piece. The software tool for managing it is mostly all about sort of what it is designed to give me command to help me know that when I'm about to do a put back that I am merged up, I'm synced up with the current code base and within the tree my versioning is correct. It provides the infrastructure so that it locks down the tree while I'm doing my delta change to it. It provides tools for the gatekeeper and such to be able to do fix up operations like pulling pieces back out, pulling versions back out you know, delta changes back out if there are problems with them. It provides some hooks and tools for doing checks for whether or not you have, for example, had RTI pools done. Whether you cut your, what we call, style-checking and that kind of stuff, so it's bunch of hooks and tools which will make sure that everything is correct before you put that code back in. The tool set if really all about making sure that when I am ready to put a delta change back in, that delta change gets in the with the current train and it slides in appropriately and doesn't collide with someone else's change. Mostly what the tool's all about is like what you described, you know, when you asked the question about what happens when someone else will try to [put in their] change when you are trying to do your change, [it] coordinates that. And on top of that it makes sure that I've crossed all my t's, dotted all my i's before my change went back in. So that's what the tool really does.

Are there any external entities that influence your process – e.g. FDA? How do they influence the project? How is this influence handled? Do they affect the design process heavily? How about the testing?

We do have some. Probably the most significant influence on the file system specifically that's outside of Company F is the standards body, POSIX. The UNIX standards. Just because Company F likes to try to produce standards conforming file system, so we try to maintain those standards. That is probably the only really significant external influence that strikes me immediately.

Well, what happens is that they tend to impact our, not so much the development process itself, but the design aspects of it. So when we are doing a feature design, for example, we'll have to make sure that we're standards conformed in terms of interfaces. It's the interface level that will impact us. So they are pretty minor impacts are the most part, although it can have architectural impacts or influence because standards do take you about certain things and then folded to very fundamental positions on our parts but well ok. So I understand that we have to provide this information in this way and therefore we have to do it this way in the code and that sort of thing. And then what happens sometimes is that we'll say 'we can't possibly do it that way' and then there's this whole process we'll have to go through where we'll have to go back to the standards committee and say 'we need this special change to standards because it doesn't make sense'. They're written for us to be honest [its like a] riddle written for UFS. Its something totally different and so first time it doesn't make sense it goes back to us. So that will change.

What other tools do you use to assist your development process? Why do you use them?

Source code control, yes, we use that to key us. The model that we use is a model based off of a gate which is our code, stable code, and then when you want to make changes to it, you take a child of the gate then you do the development within that child and they you put back your changes back into the parent. And so that is the general development model in terms of code management. You are taking a snapshot of what's currently there, making changes to it, and put the delta change back into the parent. When I'm [going to] [fix] a bug, I'm a child of the gate which is the operating system, make my changes in my own copy of it, and then put back my changes. Outside that in terms of writing code and that kind of thing, that's just pretty much standard editing tools and that's engineers choice. VI by far the [favorite] tool for doing code development.

How do you coordinate with different teams to pick the right build number?

So its usually a matter of, a combination of it, usually its something also small that your doing that usually not very much in coordination because no one else will be working in your area. So and if anybody else is, its your area, so they need to coordinate with you so they have to ask you pretty much. If on the other hand you know you're going to be reaching into other areas of the OS that isn't sort of your area but if you put the change back it will impact something in the operating system that isn't your project then I will usually go to that group and say 'hey I'm [going to] make this change and put back in this time frame and I need to coordinate these changes with you' and I'll usually fess up pretty early cause they may want to have a say in the change you are making in their area. Typically its required informally but you'll get in trouble if you don't do it but its pretty much required that if you're [going to] change in an area which is not yours, so if I'm [going to] make a change in some other project I need to make sure that I have that code reviewed by members of that team before I can put back so they'll be well aware of what I'm doing. So I'll will be in contact with that team and I'll have them look at my changes and if they have problems with their build they'll have to come back to me and say 'this shouldn't be happening'.

What kind of development - testing ratio do you have? When does your testing phase fall into your lifecycle?

Its not enough testers, typically. But as I mentioned, we have about 15 developers. We have about, I think we have about 6 right now, full time testers. So its you know roughly 3:1 developers and testers right now. And that's pretty good. We have some teams in this company who have very little test support. Sometimes we have none. So whoever's in there they'll have to do their own testing. We have a test team and they are devoted to us and that's really useful.

C.7 Interview G

Starting off… can you tell me more about what you do? Your problem domain? What issues are you trying to address?

The problem domain is that we develop these middleware solutions to something called electronic program guide. You are familiar with DirectTV or Comcast, basically it's that front end that users see when they watch TV. The GUI that allows them to select programs, allows them to set recordings, so its gives the user experience, heightens the television experience for users. So I'm part of the group that develops the middleware solutions for that problem domain. Middleware meaning we allow the vendors the ability to define the applications quickly and get them out to market more. On the lower level you got the set-tops and the hardware, our software is right in the middle that abstracts a lot of the hardware, equip it with the set-top and then then there's the vendor, somebody who comes along and says 'hey, I want to program this guide and I don't really care about set-tops.' So that's what our company does, it provides the middle layer. We're providing a lot of API and the infrastructure behind that.

Do you directly interact with your customer?

Sure! Sometimes they say that 'We want the guide to do this', 'we want the application to do this', so in order for you to do this you have to take the lowest level into account. So they may say 'well we want it to do this', we may come along and say 'no, you can't; it's too much. Its taxing the CPU, you may not be able to do that.' So from an architectural point of view that's what our software has to sort of act as a buffer between the lower level and what the user perceives. Obviously, when the user presses a button, he wants instant gratification. That may not be possible because of the box they are using, but of course the user doesn't understand that – nor do they need to. But we act as a buffer sometimes; we have a lot of interactions. Requirements – we are part of the requirements process because obviously if the customers say that 'we want us to do this within 2 seconds' and it's not technically possible, we need to be able to tell them right at the beginning 'yo, we can't do that.'

How do you facilitate these communications between you and your customer?

Like everybody else, meetings – constant meetings, on site meetings, Demos. They may say ‘well we want to see it anyway’, we say ‘well we can show you what we can give you’, so demo is one of the quick ways we can facilitate these things, get our point across.

How big is your team? What are the different roles on your team, e.g. Testers, Developers etc.? What kind of interactions do they have and how do they have it?

Right now about 30 developers and about 5 testers, and 1 program manager. So the ratio between the developers and testers is 6 to 1.

As to interactions, it depends on what part of the project you are on. In the beginning - not much. We’re off busy writing those requirements, and they’re off busy writing their test plan. We’re at the part of the project now where it’s a sort of mature development so we have a lot of interactions with them now. In the beginning we didn’t, in the beginning we were off developing our requirements and they were off developing their test plan. But now, the process more like they are looking at their test plan, they are looking at procedures, they are testing the software and they are coming back with bugs written against the software and as developers not only do we have to fix those bugs, not only do we have to implement whatever we have to implement, but we have to fix those bugs along the way. So in this part of the process, we have a lot of interactions.

What development life cycles do you employ? Why do you use this lifecycle for your team? Does your team size or structure or project needs influence the lifecycle?

Probably waterfall, traditional. Requirements analysis, documentation, coding, unit testing, integration.

How long does your project last? Or how often do you make deliveries to your customers?

It’s tough to say, I would like to say it lasts every month, basically it can last for a month, or it can last for a week. For example, there’s a critical bug and it needs to get fixed. We may release something that we just released a week ago. Get everything stable, maybe once a month, but the norm’s usually 2-3 weeks to release something. It depends on when the bugs tend to pile up and you want to release something for.

How do you manage these deliveries? How do you track the timeline such that it is finished within the scheduled time? What do you do if the schedule is not met?

That’s what the program manager’s job is. At my level, at the developers level, it pretty much this way. The program manager, in conjunction with the customer may say ‘we need to do a release because bugs 4, 5 and 6 are being addressed here. And the customer wants it now.’ ‘But we did one last week’, doesn’t matter. So here in this company, we have typical releases that are final release, we have alpha release, beta release. Those are the big milestones. In between we may have between 1 and 30, depending on what bugs are part of the release. But it's the program managers job to manage that.

Can you describe the workflow for each release/iteration you define?

Well the requirements come in. First the program manager looks at who is available to implement them. How long it's [going to] take. Is it part of the requirements, is it an add on – these are the questions we debate on. The cost of the contract, what are we getting for it, how much are we getting for it, when do they want it, do they want it at the next release which is a month from now or do they want it in a week. If they want it in a week then who's available. Once they start getting those preliminary things together we'll have a meeting with the team, the team will have a meeting with the program manager saying 'hey, here's what's come down through the pipes; from the technological point of view what do you guys think, can it be done in whatever customers said.' We go round in round, maybe, maybe not. More often than not, the customer always wins. So even though if we say it's [going to] take us a month, the termination is usually 'yeah, but we need to do this within a week, what can we do in a week's time?' 'How much can we do in a week they want you done?' Developers will go off, trying to code the thing and a lot of times doing their unit testing in whatever time, give it to QA to test, QA blesses it, and then program manager determines 'okay, we release it.'

In this workflow, do you use tools to enhance the process? Where? What are they? What do you use them for? Why do you need them? How often are they used? Who uses them? Have you ever run into problems using them?

During our status meetings within the group, so we have a group meeting with the program manager and sometimes within the development team itself, we usually outline, okay 'these are my requirements, this is what I'm doing to get done. I'm creating a new module', so we start discussing the technical aspects, 'so here's the module and we're [going to] start implementing this', 'here's the module, we're going to implement that'. During development that's our main way of tracking requirements. When we finally do unit testing, we usually have a table of the requirements and right next it to it we document how we satisfy those requirements in terms of what modules we wrote, the unit tests we used, under what conditions we tested them in, and if it passed or failed. Once we've done that, the developers anyway, that's it we're done, we can give it to QA and let them check it against their testing harness. So from the development point of view we don't really have a formal way to testing these requirements to prove to me as a program manager that you satisfy these requirements. I think the philosophy in this company is that we rely on the test people to bless that we have done the requirements.

I've seen developers use excel document to word document, pretty much a table to literally lifting the requirements, copying it to a word document, adding their stuff on, and comment.

The tools always have to be balanced with the time constraints, I can only speak for this industry, in this industry we want ‘it’ yesterday, but I’m probably sure it’s the same for everybody. So I think the main focus is, get these requirements, get them in the system, develop them, do the unit tests, and then give it to QA and let them do their formalized testing. And I can’t speak for what QA does, I’m not in QA, but they have their own process.

What kinds of tools do you use during your implementation phase to assist QA and reliability of each delivery and testing? What languages do you use for development? Do you use an IDE for assisting in the development? How effective do you think it is?

We use processor specific tools. We have three or four set-top boxes, each set top is a specific CPU, for example we use MetroWorks, WinRiver is another one we use. It’s piped down to the processor. Whatever IDE we use, it depends on the processor.
All our development is done in C language, unit tests as well. And Motorola assembly and whatever assembly the chip supports. It’s very specific.

When does your testing phase fall into your lifecycle? When do you make deliveries to QA and how often do you make them?

We don’t have a formal process. QA is constantly testing all the time. Once we give them an alpha build, the QA starts testing, release 1, release 2, release 3 etc. Once in a while they may do some regression testing also, so it’s not like we develop, we give it to QA, let QA do its thing and we wait for QA to come back. It’s always ongoing, we’re always developing, we’re always either fixing more bugs or implementing new requirements for the customer and at the same time QA is always constantly testing in the most current build.

You mentioned different build types, can you describe more about it?

Sure, Alpha build is that first build we give to our customer, very rough build saying that yes there are bugs in the system. All the requirements may not be in there but we give you the basic functionality of where we are going. That’s our alpha build. Our beta build is basically is that build where we’re 90% done. We acknowledge there are still bugs in there, we acknowledge that there are requirements that may not be in place. Those don’t have numbers, they are called, alpha build 1, alpha build 2, beta build 1 beta build 2. Once you start giving them numbers – release 1, release 2 – that’s when we signify to the customer that this is production quality build. So right now we are past alpha builds, we’re past beta, and right now for our main customers for the project I’m on, we’re on the rolling release 1.1 depending on whatever bug, whatever requirements have been implemented.

Do you communicate with other teams? What communications do you do with the other team? How do you communicate?

No, mostly within the development, program management and QA. That’s it. We are 30 people working within the same problem domain, so I guess in a day to day, yes I’ll talk to them and say ‘how did you do this’, we don’t have a formal, our resource sharing within this group.

How do you gather your requirements? How do you manage them?

On the spot, the customer came out with their dervities spec, and basically what it is, is a very vague ‘this product shall do that’, ‘this product shall do this’, with no culmination to the basic structure to the basic technology and then our program manager’s next task is to take that through meetings with the customer, through meetings with the architect, and the development, they try to refine that ‘okay, so the derivative specs says that we need to show this page’, or ‘display this pop-up’, okay, ‘where, what color, how often’ those are the things that the program manager in conjunction with the architect will now start to refine the process, ‘how often when the user presses this button, is it this pop-up or this pop-up. The user sends out comments based on his derivative spec, he also releases alpha requirements specs, beta and then a formal release requirements document 1.1 and we send them out for comments to the entire team and the developers and the architects will read the documents and start putting together their comments in there. And the commented requirements is taken back to the customer, the customer will then re-review it and its constantly going around back and forth till we finally get this document that hopefully will be well enough to write requirements against.

We used email forwarding, meetings and we use the comment feature in word documents that’s what we use. Once the requirements are complete, we store it in a formal directory. We have a formal file system directory structure; we have technical documents, program specific documents, technical etc. Each group may have their own directory under there, they store design documents, and we have a central, a file. And we put all documents in CVS.

Do you use object-oriented design? UML?

No, my group does not because we are at the microprocessor level. Another group that writes the GUI tools in C#, and they use the OO and UML, class diagrams. That group does use OO technology.

How about design tools?

The things we use most are Sequence diagram and our own version of UML, you know the class structure, since we are not OO, and the class structure is more like a function data structure type thing. We sometimes use relationship models to figure out which function’s using which data. So, we don’t have a formal way stating you must use this tool to do this. We don’t have that.

How do you perform testing – Manual or Automated? If it's automated, what tools do you use? Are they off-the-shelf or custom made? If custom, why was there a need to create a custom tool? – was there no equivalent available on the market? For manual testing, how big is the QA Team? Is this size good compared to your development team size?

Both, the testers have automated scripts. They have this mechanism where you can put a remote in there and it'll punch the keys so that's an example of automated testing and they do manual testing, so they literally go requirement by requirement, so the requirement says its [got to] do this. They push the button, did it do it, yes or no. Those automated tests are usually 3rd party tools. We don't develop any tools, the QA doesn't develop any tools. I don't think in terms of software they got mostly 3rd party stuff.

How do QA and development communicate with each other? How are bugs tracked? How is the fix communicated back to the QA? Do you think this is the best method? Why?

We don't have a formal schedule like at this time the QA has to get together with development team and talk about the problems; we are constantly talking with each other again just because our process is ongoing – we make a release and they are always coming back with something as simple as hey 'I just found a bug, coming your way.' So they enter it in bugzilla. The developer whoever is responsible will get the email and they'll talk to the QA and say 'ok, tell me more about the bug how did you duplicate it, you did you get it.' So supposedly, in bugzilla they will have a detailed step by step instruction on how to manifest the problem. Sometimes, the developer who's doing it can't duplicate it so they talk to them and say 'show me what did you do it?' But we don't have a formal where we have to get together and talk about that. However, if it's a critical problem… our bugs are labeled show-stopper, high, medium, low and then documentation. We have 4 levels of classifying bugs. A show-stopper is exactly that. Everything grinds to a halt, in which case the QA team and the developer team might get together in the room and say hey we got a problem here. So and so found this, what's going on? You guys developed this; you tell us what's going on? Okay, it involves a fix. The development team may say 'we need to move this, this and this.' Then the QA team may say, how do we test this now, how do we know it's fixed. So, for something like that there's a formal way we might communicate, again through meetings. Mostly just ongoing day to day, there's a bug kind of thing, it's coming your way, look out for it.

Are there any gaps in the development process that you think you were not able to address because of a ready made tool, or because of the project/team characteristics? If so, how do you address them?

Yeah, but here's the thing, most of the time you don't have time. Your customer wants it done because competitor X is releasing his next week. And as developer you always want to be able to say, okay, you need to think this through, that's ideal. In reality what usually happens is in our field, we might say, ok this part of the code there’s something similar to it. So we tweak it. So we take this thing and change a little bit, or this project way back when from another vendor [sort of] do the same thing, can we take that project, parts of that code and tweak it so that we can have it on here. But time is always, it’ll always be nice to talk to the customer and ask him what you really want, is this what you really want? And I think, it’s kind of pessimistic thinking, but 9 times out 10 the customer doesn’t know what they want. They pay you to figure that out. And 9 times out of 10 when you figure it out for them, it’s not what they want anyway. Yeah, in reality all the tools in the world can help you out. The always have to give away to customers, at least in this industry. It might be different for medical device companies where you have FDA and other form of processes.

Are there any external entities that influence your process – e.g. FDA? How do they influence the project? How is this influence handled? Do they affect the design process heavily? How about the testing?

The competitor. The number one, biggest external entity. It’s like ‘WOW! These guys came out with this wiz bank application and we don’t have that! What’s it [going to] take for us to have it?’

Our main competitors are always trying to outdo each other. The government is another external entity. Whatever we do, we have to within our own (SVP?) because there will be standards that the government enforces. The biggest requirement we have to abide by or build into the system is EAS, the emergency alert system. From the front end, it looks pretty simplistic, but in order to deliver that from the back end there is a lot of effort involved. So the government is also another one.

Do you have a continuous integration system?

We have a dedicated release engineer. Hi job is number 1 to maintain the build integrity of the environment. Maintain the script, it’s one of his main roles so that we as developers have to build to test our stuff; we are not banging our heads against the wall because the script broke. So it lets us concentrate on our stuff. So whatever process he has that’s the build environment responsibility. So he starts the build at the direction of the programmer ‘hey we need to do the build and give it to the customer this afternoon. Can you do it?’

We do a nightly build so that we can assure ourselves that nothing broke. That’s our one main reason to have nightly builds. So that if we do have to release something really quick we know it’s there, we can build it right away.

What about the code review process?

Code review usually happens during status meetings. The team lead might say ‘how’s it going’ and again this is the black art. Software development is black art, we might say are you 50% done, 75% done, so there’s always the information that’s being conveyed back. Once the developer says I’m done, I’m ready, the team lead might say, okay when do you want to schedule a code review. In that code review, the entire team is not invited because they have so much to do. We might select 2 or 3 people to sit down and you might want to look at the code and review that way. So prior to that we have the design review, way back when also. When we get the requirements, we sit down and say, how are we [going to] do this. So…

What other tools do you use to assist your development process? Why do you use them?

We just use smart CVS for source control. Cause its free. Besides debuggers, we AraxisMerge, it’s a big one. What that tool does is take 2 versions of code, compare it, highlight the differences between the 2 and allows you to merge back and forth between the other. Again a lot of the stuff we do is something once we get past the initial nitty-gritty of building something to round up its merging parts of the code. For example, again, we just have this requirement that we had in this project way back when we did that. So we take part of that code and merge it into ours, AraxisMerge allows you to do that easily. It’ll highlight code differences, merge them back and forth. We use Winzip. And we use these tools every day, its part of the development. You cannot go on without them.

How formal or informal do you think your process is?

Our requirements are all external, we are pure software. We don’t develop a shrink wrapped product that Microsoft develops for example. Our revenue stream is based on customers. They are big and only customers. In this industry when they say jump you say how high. In a medical company, if we have release something next week, people would say ‘whoa, we have check with the FDA and all that’. The biggest joke we have here is ‘what’s the worst that can happen if we release buggy software?’ The customer won’t be able to see his TV. Big deal! We can do another release tomorrow. If you release a defective file system or a pace maker. So the consequences there are not equal, and maybe that’s why we don’t have much of a formal process like ‘OK, we have to get to these right now’ or so on. Whereas in other companies you have to go step by step and there is a defined process. In medical companies the government has a hand in defining their process; in our company we define our own process. And if we don’t follow it, then we don’t – there are no legal ramifications other than we’ll lose business. There are legal ramifications if medical companies release software that’s buggy or buggy product. And that’s an overriding concern and I’m sure that’s why such companies take their time doing what they do.

Did you have to freeze your requirements and design in a waterfall method?

Sure we freeze them, but then the next day, the customer says that we want this, so we unfreeze them and we do CCB, change control board, we review what the changes are, where the requirements are frozen as is. And usually what happens is if the customer says ok we want this added on, the company has something called a TCB change review board. Not only do they get architecture locked, that's when they get the program manager, the sales and the marketing, cause now you are talking about ok, what's the cost of this thing. Because it's part of the contract. So freezing requirements, yeah it happens, but again it's not like you can wave it at the customer and say 'sorry you can't do this, we're frozen'. Cause the customer would say, okay fine we'll find somebody else. So we try to work with them as best we can, we accommodate added requirements. The biggest frustration in this industry is Feature Creep. You've frozen the requirements, the customer is satisfied, the application does everything they wanted to do. A week from that they might say, 'it'll be nice if besides doing this it can also do this'. So you start off with this requirement and they get bigger and bigger and you get what we call a feature creep. How that manifests itself is when you've developed to a certain requirement and now the customer wants this and you're sitting here and go this is not [going to] work! Our design doesn't support it. It's a vicious cycle, but in reality they want it and you have to give it to them, and a lot of times you hack it in. And from there you got the bugs.

How do you document them?

We release a new version of requirements document with these features added in, that's all. Our requirement docs have versions on them.

C.8 Interview H

Starting off… can you tell me more about what you do? Your problem domain? What issues are you trying to address?

I am an engineer here. I work on the cell phones so, what we basically do is that, we have the shrink wrapped version of the commercial phones, so my job here is to make sure that there is a platform always available for the developers to test and work on. So I get them the latest chip set releases which come from the target team, the different chip-sets provide different releases. My team is getting the latest releases from these teams and porting it to the software that we want which could be used for certain module updates or removing certain modules or inserting certain drivers. So at the end of the day, basically our concern is that we do have a platform ready so that the testers and developers can work on it.

Do you directly interact with your customer?

Yes. I'm kind of like a point of contact for the entire platform that we create here. The customers are the people who work on this platform, the customers are internal, so I do have direct interactions with them. Also, some of the projects that we work on directly deal with carriers. And they have some requirements and our work here is to make sure that the requirements are met. And whenever we meet, it's like whatever they want we have to provide in the build on time. So we would need a lot of coordination with the customers and our company to get that done. The interactions might not be on a day-to-day basis. It's not on phone call basis. But definitely over emails; if it's internal, for internal customers it is on phone call basis or they can just drop into my office or they send an email, voice mail or any of these. For external customers, it's highly unlikely it's going to be that informal. There is a system in place that would (SR system). It's a database where they put in their queries and we get their queries, it's a very formal way of tracking the issue, but with some carriers they will want things done pretty soon so they just shoot an email. This SR are the Service Request, it's a good way of getting to it, but it will take some time because they put in the Service Request and then the project manager will look into it, analyze it and decide who it should go to. It's a little time consuming. But in it, the customer knows exactly who to talk to, for some of the issues, they just know that I'm the point of contact so they just send me the email and I respond to it. But it has not gone beyond it; we don't do it on the phone.

How big is your team?

My immediate team that I work with on the project is just 3 of us. And as for the location team we have 20 of us. And overall the entire platform that we are providing, I mean we are providing a software layer on top of which any developer can develop their apps, that team would be like 500 some people. It could be bigger as we are hiring like crazy. So there are different levels, typically I work with the immediate team on a day-to-day basis and the people with whom we interact is at a different level. Then the overall project is at a different level.

What are the different roles on your team, e.g. Testers, Developers etc.? What kind of interactions do they have and how do they have it?

On my team, since we are providing a platform, we get in a platform, we do sanity testing to make sure its kind of stable; do the porting process and while porting, if something is missing, then we do the development of it, if some API is missing, some module is missing, then we have to develop it. And once we do the developing, we also do fixed sanity testing and send it out. This is the responsibility of the team I am in, but then my role is not very typical across all the teams, but I mean its kind of a job where we take a platform and we're responsible for the entire platform, you cannot slot it into any category like developer, tester or porting or build and integration. It spans across all these things. Previously, I joined this project like six months back. Before I was working for another project which was very development oriented, there was no interaction with customers, I would just do my coding and I would interact with my project manager who interacts with the customers. So in that case, the project was, there were 3 people working on the project, there was 2 testers, and testers come into the project when you are like half way into the project. That's for the smaller tools. But given an entire platform, there's constant testing going on all the time. And if there are like hundred developers, there would be 50 testers. So 2:1 ratio.

What development life cycles do you employ? Can you describe the workflow of your project?

So, basically what I would like to say is that this company is a totally customer driven company. So we get the requirements from the industries out there. What the customers want, what the carriers want, what we want, so that's how we get our requirements and then there is a very strong design phase that goes on where we sit together and drag our brains trying to come up with all possible scenarios and all possible cases where you know, just to make sure we have accounted everything. And we also account for things which we might come across in the foreseeable future. So, but then they will be low priority, so at that point we know what our requirements are, we know what we can get done and we have to prioritize. So we do prioritize all those things as to what exactly is important right now and then we prioritize it in terms of releases, so let's say for 1.1 we do this, for 1.2 we do something else so once we have are done with that, then the resource thing will be done. Like we'll have to plan how many resources are needed, beat hardware, beat software or beat people, beat testing, we have to scope out everything and then we make the project plan and try to follow that. That's how we start. And then we go back, and there are certain things you don't know until and unless you start working on that. Maybe the hardware is not available and the chip which had to get taken out, got delayed, so have to move to a different target. If we have to go to another target we have to develop something else and we have to take somebody else's help. So we have to go back and change the requirements and design again and work our way through it. Or in my case, we are totally customer driven, so the customer can come back tomorrow and say we don't want this we want something else. So they can go back and change the process. So this is [sort of] like spiral/ iterative development, we don't follow the waterfall process where you do all at once. I don't think you can ever follow that in real life because, at least working here I feel like, there are so many external factors that come into being no matter how smart you are and how many things you take into account while designing this, you are still in the hands of your customer, let's say your customer says no I don't want this here, we want something like a touch screen, and we say 'oh, no, we can't prepare a touch screen' then that's it, tomorrow you start working on touch screen since its more important now. In that regards we don't follow a waterfall model, we follow this spiral/iterative development where we keep on evaluating what we are doing and depending on test requirements, depending on what we can get done, we plan backwards and because its just not developed in the right… cause the development has to be done way in advance – there's something called feature complete, code complete and test complete. Feature complete is where you have all the interfaces defined and ready out there. Code complete is where your code is done. That's a different stage. And then code freeze is where you cannot check-in anything. I mean difference between code freeze and code complete is that in code compete, if there is a bug, you can still keep checking in the code. But then you'll have to follow it up with a credible CR. And they have servers, believe it or not your phones have 20-25,000 files and when we have people working on so many files we need to coordinate it somehow. So we have these centralized servers, they are called version control unit so what we do is that we lock up these servers when you get pass the code complete and then any check in you have to make into these servers has to be followed up by a valid reason of why you are doing it. Then you have the code freeze and after code freeze it goes to… and even during this time, ever since you are done a kind of a working model testers get involved and testing goes on and testing gets very rigorous when you approaching code freeze because that's when you start finding bugs, that's when you keep late hours and fix more and more bugs and pushing the code so that you meet the deadline and once all this is done, we give it to the construction team and they pick up the product and they'll have to spruce up the product because there are so many internal things which is in the code and which we need to clear up, so you clean up the entire code and then send it out to customers. So that's how it starts with the customers and it ends with them.

Do you experience cost or delays to your schedule when you run into the factors that cause you to go back and re-visit the requirements?

Regarding the cost factor, I don't deal with that level of scheduling, so I don't really have that responsibility, but regarding delay yes. But again, you know since we are in a stage where we have just started working on it, though its delayed we can negotiate in terms of features which go in. If you promise something else we say that 'no if we make this change it starts going with this in this timeframe.' So we will negotiate in terms of the number of features which would go in. I know it's a pretty bad programming practice to put in more people in the end, so we don't do that. We avoid doing that. What we do is, we rather negotiate and have some features totally ready than have many features which are buggy.

How long does each release last?

In the whole life cycle, it usually is anywhere between 6 months to an year. Initially its totally customer driven as I told you it was whenever we needed… it was that way like long back. But right now we have internally daily releases to the testing team. And once we have daily releases, once we make the target it ready, we release it to the internal customers for alpha testing. And once that is finalize then we release it out to external customers. And that is one in a year or 6 months and then there will be different phases of the same thing, I mean there will be different patches. So if things keep going on in parallel all the time so they have deadlines every month so each project release depends on the scope of the project. Its like if we're releasing the first initial release of the project then it will take at least 6 months to a year. If you are making incremental releases like patch, it will not be more than a month or two. And if you are dealing with other incremental releases you can do it under six months.

How do you track the timeline such that it is finished within the scheduled time?

Um, we do this in Microsoft Project. So we right from day one, once they have project plan ready, once we figure out what resources we need, we put these things in MS Project. And we have a project manager who takes care of these things. They take the help of a program manager so they have a project manager and a program manager. The project manager is specific to our particular project. Program manager is more general, they deal with just the resources, the hardware, the software and all these things. So they project manager what they do is they keep track of our schedule, they keep track of how long it takes to fix a bug. I mean they have it totally scheduled, so that's how they keep track of it. And more than that, it doesn't happen that at every meeting you don't have this project manager sitting and asking you all, why you didn't do this and why you did this. That doesn't really happen here, there's a project manager somewhere, we don't really see him everyday we just know what we need to get done and we have these regular meetings and we know what we need to get done by the end of the day. So other thing why we feel it is pointless to follow schedules is that we give the status report every week, like I send the status report to my boss and he sends the status reports to his boss so he coordinates and sends it to the VP, so its like there is a progress report that goes on every week. That's how the upper management keeps track of what's happening. And as far as I'm concerned I manage my own schedules with the help of the project manager.

What do you do if the schedule is not met?

It does happen that way, when you are nearing the end of the project. You will have to put in extra hours. In the beginning its kind of easy to go on when you are doing all these interviews. But I can't really do this when the project is reaching completion and all. So, at that point, what we basically do is, from a developer's point of view you need to let your management know about this and then once your project manager comes into picture he will take that into consideration and he will push back the customers saying that 'We will not get it done, we will need some more time to get it done.' I mean the whole point is that you feel free to push it back because there should be a balance. When you feel that things cannot be done, we should push it back.

In this workflow, do you use tools to enhance the process? Where? What are they? What do you use them for? Why do you need them?

Basically, tools in the sense we can't do without our task centralized servers that I was talking about like version control unit, that's something that you cannot do without. And I think that's taken for granted. So that's fine. And then there are certain bug tracking tools. Bug tracking tools are pretty good and whenever there is a change in status, it will just let you know that there is a change in status and you can just go in and see if something has changed, if someone has fixed the bug somewhere, it will just go and put in the change somewhere. Other important thing that we have is the internal communicator. It's like a yahoo messenger, but it's a communicator project by Microsoft, this is really useful. And since we are in a remote office, we deal with our headquarters quite a lot so we have VNC, so that they can VNC into our machine. Or they can through Office communicator; it also provides a way of sharing your desktop. And we also, its proven very useful because I have been managing a few people in another country and its working out pretty good using these tools. And of course…the phones.

How often are they used? Who uses them? Have you ever run into problems using them?

VNC and anything to do with Remote desktoping we need it on a day to day basis, I mean lets say there is a bug. And you have seen the bug in somebody else's court and you have no way of fixing that, I mean you can fix it, its not like you cannot fix it, but instead of you spending 8 hours on it, somebody else can fix in 4 hours. So let's say you are in Bangalore so you need to coordinate a time so that he can log into the lab machine or he can VNC into the lab machine, so what I'm saying is that there are certain issues or bugs will occur, you will be on those machines all the time, but that doesn't even happen until and unless you start debugging. So anything dealing with remote desk topping that's not that frequent, but anything that deals with interacting with people, I use communicator quite a lot, its very useful. You have email but communicator is like very useful, very informal way. Its just a good way of going to a somebody's office, in the communicator I just ask them 'Are you free now, can I come now' and he says 'yes' so I just go to his office. It's a combination of all these tools, its not like we use one tool the most or …

Do you communicate with other teams? What communications do you do with the other team? How do you communicate?

Right, basically, that's what my team's all about to smoothly have coordination with other teams because we get their product from somebody else, we port something from somebody else to our target. So yes, I deal quite a lot with other teams, very much. And again, one key thing here that I have not learned in any books is that when you are dealing with other teams you just need to use your skills and find the right person to talk to there. Not everybody will be receptive to all your burdens and all your problems, and the right person, keep bugging him. And this is something that is on a team–by–team and case-by-case basis. So regarding the communication with these teams, some people are very comfortable with the IM, Instant Messaging and some people are not. So in that case, just email them and hope that they respond and we also have pagers, cell phones and of course, office phones and lab phones which they do mention in their PH. We have this thing called PH. PH is the area where you have information about the employees. So we have all these numbers like lab numbers and cell phone numbers and test numbers and then we can look up their name in the office communicator. First of all is email, if you have to do something or send something to a group of people, email is the best option. If you have to talk to just one particular person, no where that particular person has answered and its not that important, IM is fine.

How do you gather your requirements?

Gathering requirements is again, we have to deal with customers, so first thing is the customer comes back to us saying 'Okay, your 1.0 is good, but in 2.0 we want something else'. So that one immediate thing which would drive it. We did have to give all 2.0 features, but then along with 2.0 they need a lot of internal things with few new features, so we can take, for the tools or the platform we have meetings with these teams. We have bi-weekly meeting with the stake holders of our products who would come and discuss the features. And depending on the requirements we would make changes against what we were doing. So it's like we would constantly keep in touch with them. We would get rough requirements in the beginning but even as they start working on their product and as they are dealing with our product, if we constantly keep in touch with them maybe we can do something to quickly start using the product. Or maybe if we delay doing something, they can come back and say, yeah that's not important we can start working with something else. So we constantly keep in touch with the customers – at least bi-weekly.

Even with the external customers we have constant talks with them, very frequently to figure out what we need to do and constantly use our iterative design process. We can design for change, its not the source where you can design for change, so you make it in such a way that we gather all these requirements and we get the requirements, we try to develop the code in such a way that can be enhanced, headers can be expanded, So…

How do you manage them?

So, in the requirements phase its more like…see we have the requirements that translates into something called the requirements matrix. And this requirements matrix translates into another a Microsoft Project plan and this project plan, it could be Microsoft Project, or it could be a Wiki Page or it could be the set of slides. It depends on how your project manager is managing it. So lets assume it's a Microsoft Project plan, in that case, once you have the requirements matrix which is translated into a project plan, you can keep track of it there right? Yeah, each code, again coming back to people, if you have independent resources, and these are the people working on this, and these are the hard ware resources which are needed, and these resources will not be available till the end of this month. These people will not be free until the end of next month so all this planning needs to be done in the plan.

How do you manage traceability between requirements to design and design to test? Do you use tools to manage this? If yes, what tools do you use? If no, why?

We have the requirements matrix that I was talking about and from the requirements we develop the test cases, so when they run the test cases on the design your tracing your requirements back to test, back to design. See, tests are driven not by design but by requirements, and design is driven by requirements so both of these are two separate entities but both of them are designed by requirements, so when testing is done on the code, I mean it just need not be code, I mean you can have design reviews right? All these reviews, when you have a design, there's an enormous design review that goes on, it goes on for month right, sometimes they are still running. We spend so much time in designing but then its very important to get the design right, so yeah, that's how we drag it back. It could be design reviews, code reviews, that's the code part of it. Here testing is totally an independent part of design so…

What languages do you use for development? Do you use an IDE for assisting in the development? How effective do you think it is?

Its basically C, C++ to certain extent, a working knowledge of assembly level programming is really very important. We use it to debug the code so you need to understand how it gets translated on the device. On top of that if you have some windows programming we don’t do much of it, but my other team mates do work on it, but primarily its C.

We do not use… okay, if you are working at a higher level, we would use Visual Studio IDE. If you are working on embedded system device, like I do, so I don’t choose any IDE’s but I use editors like SlickEdit or VM or GM. And moreover most of the setup is already done, its more like you have all the tools set up in such a way that you just open the command line and do it on the command line because they, I mean, most of us prefer using the command line. So… we just use the command line to fire off a phone build so.

How do you perform testing – Manual or Automated?

It’s a combination of both, like I told you, I mean we have like 10,000 test cases or something like that, so most of the testing on the day to day basis is totally automated, wherein I start the build and that makes the build load on the device and tests the modules and starts the testing. There are things which you cannot totally automate, for example, if you have to touch something to do a street test, we do have robots doing that, but then we are at a state where we keep on changing, right now we are developing so its not set in stone. So we need some manual testing to be done. And if there are some 10,000 test cases, there will be some manual test cases. Maybe less, maybe 1000 …

And even then when I say 10,000 test cases its just the module and not the entire company test cases.

If it’s automated, what tools do you use? Are they off-the-shelf or custom made? If custom, why was there a need to create a custom tool? – was there no equivalent available on the market?

Basically, its totally perl based. So its perl and automation and Apache servers, so once I kick off an automation, it will kick off a build. So I kicked of a build for instance, and if you look at it here, pass 3164, failed 267, so it was not able to run all the test cases in this case. So you can’t really pin point and say we make use of this, since it’s the total environment. We make use of the cross compilers to compile the code, we make use of the company’s internal tools to download the code, and then we make use of perl to manage the entire process of automating and then we have some servers which will allow us to kick start these HTML pages whenever something goes on, so for instance, I showed you our version control unit, right? I was saying that there are like a thousand people working on this so whenever even a single person checks in a file, I mean, put the file back into the Perforce, it will start the process. It will make sure that that file is compatible with the entire code. Make sure that the code builds compiles, runs – I mean we can load up the device, boot up the device, once we have booted the device, the next thing would be to compile all the test cases, load them on the device, run all the test cases with the help of the automation framework that we have, it will them print out the results.

How often do you run these builds?

Regular check-in. There is no point in running a regular nightly automated build, when there is no check in. Even nightly automation is a common vice. You have like maybe a few hundred check in that go on in the morning and in the night, if you run the nightly automation, how can you pin point whose caused the bug. So in our model we start the process for every check-in that goes on. So, its totally based on requirements, so for every check in that goes on, we start the process and test it, it might take time, but we do that, we have dedicated servers in the lab, just only to do this. So all the day they have a phone connected and everytime there is a check in they just do the entire process.

For manual testing, how big is the QA Team? Is this size good compared to your development team size?

I'm not exactly sure about that…. So this tool shows that its 58 right now.

So when does the testing fall into your process?

Testing is like an ongoing thing. Since automated testing goes on every day, it's like for every check in we do the entire automated testing. And the rest of testing is totally driven by the requirements, so we started testing whenever there is a release, so we plan backwards. So the product has to be ready at least two – three months in advance, so… once we have a design in place, we start the coding then we have a feature complete code freeze, after the code freeze we start testing. I mean, testing goes on even before the code freeze, but it usually will be automation, nightly automation, check-in test automation, not much of manual testing there because we have to get the basic set of features working before we go ahead and test the more integrated features. So… once we are done with that we start the manual testing. So pretty much, I mean, as soon as we come to a point where we have feature complete we start the testing.

How are bugs tracked? How is the fix communicated back to the QA? Do you think this is the best method? Why?

We have something called TeamTrack so…we two different mechanisms; one is CRMDB, CRM database. One is TeamTrack. So TeamTrack is this tool that we use.

Its like all the CR's are here that are assigned to me, so this is like old CR's that are assigned to me (50 or so). CR stands for Change Request, if you look at it here, someone opened the CR on 12/4/07 and assigned it to me, so it also keeps track of all the CR, all the stuff that's going on. There is a history associated with it like who opened it and why he opened it and which release it needs to go it and which release it was found in and who was the tester, why did it fail, how did they test it, how do you reproduce it and the cause and effect of this thing. What happens is they will file the CR and once you file in TeamTrack, and twice a week we have weekly CR Meetings, and what we do is, all of us, all the concerned players like, I assign CR's to my team. So all the team leads of different teams will get together sit and decide what the CR looks like. Is it the device specific CR or the Platform specific CR's. I mean, we categorize CR's into different categories and assign it to that particular person. So… and at that point its assumed that Team Track is pretty complete, we have all the information that we need, like I told you right? Who, What, When, Why and How and all these things. Every information will be there but in the meeting why we have the meeting is that, we need to figure out who will assign the CRs, how long does it take to assign it, how important is it and what release it needs to get into it. Depending on that we prioritize it and set a date to the CR and once the CR is defined we have a dedicated team that keeps track of it, they just assign the CR and that CR is, for instance its CR that's assigned to me, then I fix it, I assign it to a tester or the verifier, and the verifier will verify it and assign it back to the owner of the CR whoever filed it, that person will then close the CR. These tools make it very easy to track all that.

Do you use custom tools for automated testing?

We use a whole lot of internal tools, I mean we deal with phones so we are not expecting external tools to help us develop code on the phones right? So we have our own explorers of the phones, we have our own interfaces and we have our own signal generators. Our Login and RF Generation and automation, the set of tools that we have is surprising that for testing we have so many resources, but then that's what is needed. So we have the product but what's the point if you cannot test it. So we have a whole lot of internal tools.

The only thing that we use from an external customer is cross compilers because we are using an external processor so except for that everything else, starting from how you download the code onto the device to the point where you acquire signal over the air, so all those.

The reason why we don't use external tools is that it's a totally proprietary software that we do and we need something, and we need this feature totally driven by our requirements, we cannot wait on external customers to deliver these tools. For instant if you look at the TeamTrack. TeamTrack is very good. And its not something which every company needs, right? So this is something we can just pick off-the-shelf. But if you come to something which deals with the phones, that is something that is totally proprietary, so we cannot ask other companies to develop the software right?

Its totally proprietary and totally IP based. So we need it, and the other reason is that, our company takes pride in its testing tools. One thing is that, since we are the ones that is developing the products, we know what test cases to write and how the signals need to be generated, so we have all the IP test beds needed, so we can develop the tools much better than anybody in the industry can, so at that level, there's no point why we should let somebody else take our IP and develop the tools for us when we can do it ourselves.

Are there any external entities that influence your process – e.g. FDA? How do they influence the project? How is this influence handled? Do they affect the design process heavily? How about the testing?

We do have some defense processes going on so that does affect it but then its pretty minimal, its only for those projects, but in general, when our customer base wants something we have to get it done. Coming back to external processes, we do have something called SCM. We are SCM level 1, I guess. Basically what happens is that we have companywide meetings every quarter so in these companywide meetings the president of the division comes and talks about what the mixed priority for the entire team is going to be.

Do you have heavy requirements for the documentation process?

Documentation, yes we do have documentation, it was not there before. This company was totally an organic company which grew organically. There is no documentation when you look at the code itself, but for every new feature that we are doing for the past 2 years, everything is properly documented. The features are documented, the requirements are documented, the project design, the project plan, things are very well documented, but where it gets fuzzy is when you actually do the implementation you'll be in a time crunch. So its more like instead of doing documentation we do in line documentation. So we have very good set of files that anybody who uses the header file can look at the documentation and get it done, but as far as the implementation is concerned we do not have information about that. Its totally IP based so we do not have extra information, but when things which go out to external developers we have very good documentation. For things which are internal, its totally, we have to dig up the design more, solve the project plan and then…

How is the frequency of your requirement changes?

Its very high. Requirements change and plans change, so its very high.

C.9 Interview I

Starting off… can you tell me more about what you do? Your problem domain? What issues are you trying to address?

I'm the director of engineering for a small software consultant company. So we don't do products. Everything that we create is paid for by clients in need and its negotiated on a project by project basis. Broadly we deal with the flow of information in that company, trying to get value out of that for customers. More specifically its almost always has to do with document, or if not their written document, their information put in rows and columns…there are very few companies that don't use computers somehow and they pick out islands of functionality that are obvious to them that they would work better on computer. Those are always database oriented. So we rarely do that because it already was done. But there's not a lot of information other than that that's more nebulous, less structured and that where the core of our work is.

The most obvious customers tend to be the ones that have onerous processes. So if you have to deal with government agencies like the FDA or the FAA, well you got an awful lot of just documentation that you produce. We've evolved somewhat since then but that's where the business started and it's the easiest one to make an example. If you are a drug company and you have to put it in, oh I don't know 30,000 pages of documentation on a new drug, its obvious to you that having that created, controlled, searched and indexed electronically is better than having it in the semi-truck.

Our business is making their documentation processes electronic, dealing with the management and dealing with putting it in and getting it back out. The most common business case is that people tend to re-create their information over and over again if they can't find it. And publishing it back out, because a lot of times there are electronic records that are legally binding. You have to keep financials for seven years.

Do you directly interact with your customer?

Almost always. We're still an expertise so we are still expected to lead but it's the rare customer that just does 'Oh, I see you have a great idea. Come back in six months and you can deliver it.' So you're always working with the customer to some degree and in most cases, the more the better.

How big is your team? What are the different roles on your team, e.g. Testers, Developers etc.? What kind of interactions do they have and how do they have it?

We're rarely segregated into a single role, because we are so little. But generally there's a guy who's responsible for testing isn't allowed to touch development because its very difficult to test what you developed yourself.

The team is about 3-5 people. Its been as high as maybe, 8, and been my opinion it really starts to get unwieldy, if I thought, we were going to be about teams of 5-10, I'd literally break them up into segregations.

There's never more than one tester. So there's always one analyst, there's the guy who went in there to start out with and developed a lead, to me that's more of a marketer because that's where we would fit. Which is almost always me. He went it, and ask questions, so its awful lot of listening and ‘Oh, well that sounds really onerous and it must be improved’ you know, so it has that person at least part-time on field, helps the team do what we originally promised which is good. And all the rest are developers. Those developers are probably asked to do other things like tools or IT support, or write some manuals or something like that, but their job is development and anything else I can get them to do.

What development life cycles do you employ? Why do you use this lifecycle for your team? Does your team size or structure or project needs influence the lifecycle? Can you describe the workflow for each release/iteration you define?

It just doesn’t have a name. So we don’t look at our process and say we’re Agile. We’re XP, we’re Scrum, we’re Object-Oriented. Our analysis is always formal, probably more formal than you see at a company anywhere near our size and products and that’s because that analysis is essentially a contact with customers. No customer just says, ‘here’s half mil, go and do what you want.’ They expect something very specific when they go to get their money they have to write down something very specific. So we might not consider it a contract but they certainly do. It might not be legally enforceable but certainly very formal description of what have we promised you and what is the problem did we say could get better when you put it in. And that means, as you would expect, that also means that the testing at the very end of the process, not the internal testing but the acceptance testing, even if we don’t call it that because it implies that they might not accept, so we don’t want to give them the idea that they can just not accept it, but that’s what it is – acceptance testing. You know, we said it would do this, do you think it does this. Do we need to make some changes, those two parts are very formal. The development process is much less formal. It depends on how much time we have and how much you care that I can give you chapter and verse of how I don’t think formal development processes do much. But, at least for a company my size and situation, which is wearen’t very big, we use a lot of contractors but they tend to be the same guys over and over again. And in almost all cases they are people that I already knew. So I already know I have very high level strong people, I don’t have the problem like a large compare where, if you’re [going to] hire a thousand people, you know what? They’re not all superstars. Which is one of the big things why people do put in software processes. I don’t have that problem. I don’t want people to get into producing stuff that I can throw to someone, its not only makes things take longer but it’s the ethical problem of I can’t really bill the client for all the time these guys are spending writing status reports.

The analysis which partly falls under the straight marketing partially falls under taking what you learned from customers of their problems and turn it into pieces of technical projects that we can say we did this, we did this, we did this and now we know the project is done. That’s how we start. Always done on customer sites, all the eliciting requirements, talking to what they call stakeholders now, not only people who are [going to] use the system but people who are [going to] pay for the system. Its not always talking to people, but its bigger deal in a consultant company than it is for a product company. You see, in product companies, the people who are doing it probably know a lot to start off with. They don't have to taught a whole new business, they don't have to be taught about what it's like to run an airline, what its like to run a county government. Then the last part of that is done in-house. We do a lot of talking, then we come in house and generally produce the documents. That document's pretty formal, I've [got to] sign it, the customer's [got to] sign it. And that's the contract we use to decide whether we've met customer expectations. It is more formal than you may expect. Once that's done the creating of the design is pretty informal, you'll find a lot of detail in the interfaces between either components or people so what I try not to happen is have two people go off for two weeks and define things that test perfectly internally but don't work when they are together. So there's a lot of detail in the interfaces between either people or processes. I keep this one guy to write both the processes that brings the documentation and the process that brings it back out, I still want him to say how its been detailed in the middle and that's most held in an email, held in a wiki. And its held in a internal document management system. Those documents are rarely very long, but we try to make people to spend some time thinking about it. Before you just go off and write stuff, you need think about how its [going to] do it and drop it off in a paragraph in an email. That's an iterative process, of course. We have a more or less, Scrum like meetings, every Monday afternoon. Its never very long but it is, this is what you told me to do last week, they'll give you a lead. And this is what you told the rest of the you were going to do last week, did you do it? What are going to do this week, peer pressure takes care of the rest. Then generally, people are spending the rest of the afternoon creating these little paragraphs, they are [kind of] like promises of what you are [going to] do, how they are [going to] do it, just briefly. And I then [kind of] tighten the considerations. That iterates, as I say, we do one part, a week another, a week another until its like we have it up to test and then the directors start getting involved. Testing is ongoing as much it can be but its hard for me justify hiring a lot of people to only do testing. So, I've only got one real start tester. One guy, when he sends, I couldn't find anything and I think this is what the customer is asking for then it is probably true. This is not what I recommend the process goes, its just what I got.

At the end of that there's a fairly formal piece of, it probably goes about several weeks, just about everyone if they can spare the time gets up, we go to customer site, we get all the stakeholders, we do fairly formal demonstrations to people, we talk to the people who are going to maintain it about how its written. Seeing as we might know, the people who are going to have to support it, things that we might know that they don't about versions of Java virtual machines, things like that. And then the manager generally signs off the last piece of money and from then on its their problem.

Do your requirements change often?

Of course, a lot of them are failure or incomplete analysis, and a lot of this is because the customer himself doesn't know his process well enough to make a computer do it. So there was one project, we did a couple of years ago, but I was involved with this project, not with the same company but it is a good example where we went in with a fairly large team to team that repairs airplanes. They have a real around their documentation because in order to get their local FAA rep to sign off on any given repair that you have to do, they have a real formal process on how to move their documentation. And they've got engineers request in when it is essentially a non-routine repair. So the mechanic says 'I'm not allowed to fix an airplane wing that has crack more than 3 inches long. This crack is around 3 and a half inches long, I took a little rubbing, its definitely three and a half inches. I've got to get an engineer involved. And then there's a very formal process, he creates his little request in, usually they have a little folder in which they take digital folders, make his request saves, include a part of manual that's required. He walks this folder off to his manager. There are always two immediate lines between their house designs. Then that guy walks the folder and this is like something where all the high level managers are off in another building so you walk across the campus, he signs. He goes off to his engineers tell them what to make, he signs it, he walks back to the mail office and its always under time pressure because they are under pressure from the guy who owns airplanes because all the times its not flying its not making money. So, the theory in this project was, we'll make all these electronic, we'll send it electronically. We will be able to make it so that people could view and time it and the whole thing would go from an hour to 10 minutes. 16 hours to you actually put the system up for over a month. So we actually put the project in place, so that they deal with things like 'Oh, did we mention there isn't a network in' and a couple of months later they came back and we got a problem because 16 hours of the day, it works great. But they repair around the clock. And they got the guy who works the night shift, 10 PM to 4 AM I guess. So he didn't have a manager, there's a manager on duty 2/3rd's of the time. And no one besides the night shift had any idea that this is happening. So what they were doing was basically going ahead and doing the repair and getting the morning manager to sign off on it after the fact. Now the director had no idea this is happening. Or if he did, he wasn't about to tell us. So the failure in software process, it was a failure in the customer's own understanding of what the process was. It would have taken a very talented analyst to realize that there were a few shifts going on and they better talk to all shifts in there, getting sign of on this thing. So that was certainly one example. It's the most dramatic that I have on people's treat. It happens at a low level, it happens all the time, they customers say they need one thing, and really need another. So the guy you were talking was fired or moves within the organization 'cause the tester has a different idea. He doesn't realize that the process is more structured and formal because there are a lot of computers involved and they flip out and you have to adjust.

How do you adjust when your requirements change?

Two ways, if it's a lateral change, a lot of them are lateral changes. 'I want you to put this in instead of this thing' or 'Alright, you want me to change the field, change what I already put in for you but I can do that if I don't have to do this other thing.' If we can make that kind of an informal conversation with the customer then it can get handled by the developers, I get involved but generally its handled right at that level and the high level stake holders probably don't even care about because our analysis is at the business level not at the technical level so as long as we still solve the same problem, which we are or we won't do the change, then they don't care. Sometimes the business requirements change a lot, that's often a result of something that happens in their organization – they get bought, they change personnel or whatever, and that process is now back to square one, it reached that limit of parts, and if we have to ask for more money it takes very long. If they want to change its that large. Then it's a very formal process to define the size and define how much maintenance.

How long does your project last? Or how often do you make deliveries to your customers? How do you manage these deliveries? How do you track the timeline such that it is finished within the scheduled time? What do you do if the schedule is not met?

I've only seen one that went over a year. Probably 3-6 months, and 6-12 months, recently we've split between the other two, but they tend to be long. When you try changing the organizations, the basic process is pretty long.

In theory our deliverables are just code. In fact it's the collaboration of code, of it might tell how to change their process, of support when a computer gets put in, of installation of products, of teaching those products to talk with the systems they already have in place. If they already have a computer they might already have SAP, the might have Oracle, they might have lots of Microsoft products, there's not willing to change those products when a new system comes in. So it's the job of the new system to adapt to all its [got to] touch, and because we are in the business. The schedule has to bee sealed in advance, the money is pretty well defined but we're rarely in a situation where someone knows what they want so well, that they know that it has to be done in two months, it has to be done in four months. But we manage cost by doing good analysis and by keeping up dialog with the customer. Something you may be see Agile or XP and all those, you know. Making sure that the low level negotiation is ongoing, so that we don't get a nasty surprise at the end of the project. [This] is probably the biggest stopper of the project, a reason why processes fail.

What kinds of tools do you use during your implementation phase to assist QA and reliability of each delivery and testing?

None of them are process tools generally, but, we use Subversion for version control. We use several document management systems as that's kind of our business. We've a lot of them, SharePoint, Elpatchco, documentum. We use email of course. These products which are enterprise document management also have collaboration tools in them. We use that a lot in a system called DotProject, which is kind of a project that manages this Wiki. It's the mother of all process collaborator thing. But its little, it doesn't take much load of the computer so we have one dot project installation per customer, so they can kind of go in and see where we are in the project. See how much money we spent, they got the right permission to see what we are working on next. Be involved as forms, wikis they are involved in dialogs but they don't have to call us out. Not that I think they're bad, there's just be more useful testing something that touches all your business units. That would just be an extraordinary product.

When does testing come into play?

People are supposed to be unit-testing all the time. Everyone hates it. I've never seen an organization even one that claims to test like Agile, testing up front, they are writing their tests as the developers. Sometimes, to really write unit tests as you are developing at the level that the process says you do, you would spend a vast percent of your time writing unit tests. And most of them probably wouldn't be very sensible. Even if you'll be doing product, even if you have a well defined problem domain, even if the engineer know so much about the way it works that he could test the header factors. The tests the developer is supposed to do is unit testing and it is using NUnits and JUnit frameworks. I don't really ask to test every single line of code as is written, which really good testers would do and they would analyze to see if the tests would catch every decision point in the process. He's writing a space shuttle, you've got the time and money to do that and the emphasis to that. In our business we really don't.

Manual testing is towards the end of the process, as I said is, where people are supposed to start converting to, alright we told the customers its [going to] do all these things, is it doing those things. Much more high level testing, not testing every thing in the code, not even testing each unit, but our predecessor to the acceptance testing that we're going to do at the customer sites. The part where 'let's not have any surprises until we get there'. If nasty things are [going to] come, you'd expect that's where they are [going to] come up. Oh we made this promise and we forget to tell that we are not [going to] meet for whatever reasons. We assume they have java when they don't, we assume they have the network when they don't. We talk to one manager who thinks that all managers should sign something off and later in the project we find out that there's no way we're [going to] find managers who can do it. The few times that we had something come up, its always right there.

Do you communicate with other teams? What communications do you do with the other team? How do you communicate?

Where we're talking with other consultants, and its partially like a lawyers talking to other lawyers. So its rare we would get into situations where there is multiple teams trying to solve the same problem. We're not [going to] be generally part of a 100 million dollar contract with the government. Generally we would develop the lead, so we have the contract exclusively, sometimes we would hire other, rarely we would hire other organizations to do the parts that we can't but its not a very good decision to take. We're with the company's IT all the time. It was fairly common to not run the shell and get my way in things. I saw a lot of problems when the consultant companies just came in and they do things their way. And as IT you would have to support it. Feels like you passed them over or aren't listening to them. IT can make your life difficult in lots of subtle ways. So we learned to take them out to dinner, they don't have to do the work but we talk to them constantly. The implication of what I'm doing is, 'you're [going to] have Crystal Reports, can you support that? You're [going to] have java, you're [going to] have Microsoft, or I'm getting ready to decide on a reporting tool, do you have something already that I can use?' So we keep them involved. There's a steady small flow of messages one way to the manage but who ever is the money man who has us in there gets a sort of status reports constantly. Because it keeps them happy and it keeps them from doing things, not that they generally need them and not that they get the technical detail, so they never respond, but they get that constantly. Its actually [kind of] rare that there is an large internal development and I don't of a reason why, it just hasn't come up very much. I know some of these companies like the big airline and some of the government organizations have had them but we rarely see them.

How do you gather your requirements? How do you manage them?

Well we manage them in a document managing system. They are documents, they are created on someone's hard drive, and are generally iteratively checked in to start working on them. But there's no like Rational Rose style system for eliciting and managing requirements, they are just in documents. The versioning part is much more elaborate than it needs to be because people need practice. So there's work involved because the manager and they keep getting persons and then they audit the changes. If someone writes it and then they arbitrarily gets them to the manager, which is usually me, just so they can send it off to the work flows. Its not that we need to do that, its that you are putting systems at the customer's so all of us should be using that on a daily basis, so its like eat your own dog food.

Do you use object-oriented design? UML?

Not really. I was one of the real big proponents of that when I still had certain pieces on it, it doesn't adapt well to do is all. Its the object oriented processes that keeps Rombach and Jacobson ended up coming them up with, after they got trashed on, just was not work for us. When we were decomposing things as the design, we use those concepts, you wouldn't see UML except from old guys like me who [kind of] got used to using it. We don't mandate it, still wouldn't give up.

What languages do you use for development? Do you use an IDE for assisting in the development? How effective do you think it is?

We use, for programming, which is maybe be about half of the technical work that we actually do, Java when we can get away with it and Microsoft when we can't. Visual Studio, anything from 6 to 2005. Often it'll be C or C#, those decisions are made when the customer says 'hey look I have my IT, my programmers. My IT people want all Microsoft's ops or not or we hate Microsoft or whatever.' That's why that decision will be made and that's why its across the board.

How do you manage traceability between requirements to design and design to test? Do you use tools to manage this? If yes, what tools do you use? If no, why?

Broadly we don't. There's a one to one correspondence between the very start of these are the things we are going to do and the very end of acceptance tests. There really isn't a big tie between these parts of the design are meant to reduce the time that it takes for the document to move over to a manager. Once we promise them we do these few dealt with that but they will see if the cost to keep it works. My experience is been that they all [kind of] shy off in making assumptions.

How do you perform testing – Manual or Automated? If it's automated, what tools do you use? Are they off-the-shelf or custom made? If custom, why was there a need to create a custom tool? – was there no equivalent available on the market? For manual testing, how big is the QA Team? Is this size good compared to your development team size?

All the tests are mostly manual. The only ones that are automated is in testing code with one of the unit frameworks. I don't have any great ideas about testing something as great as to do in an automated fashion. Maybe we find parts of it. Message broker piece of software, we may very well fire a bunch of odd messages at it. And if we are doing that we might as well write a program that does it and you can either call that automated but really it's a manual process using the computer is all.

How do you manage the different builds and versions of the software?

Well we really don't have that many versions of the build. We use to do that when we used SourceSafe. We were more formal about that so we switched to Subversion which is a CVS offshoot. It manages versions of files very well, versions of project not all that well. But you can get to it over the web. If I had the money for a better system I would have something that was much more formal about 'yeah, OK it was first VP of text.c but what I care about is all the files that went into the build'. I used to be much more formal about knowing what customer got on a given day, on a given information and being able to reproduce it even down to the development tools, the libraries that went into it and all that stuff. I don't have that now. Its not money, if I can afford it I would fix something like that up obviously.

I switched to Subversion from SourceSafe, because you can't get SourceSafe over the web. It relies on file over locking so there are some systems that essentially try to shoehorn http access for web access that aren't very good and most companies, if you actually sit your laptop inside their firewall, they won't let you VPN out to your own.

How do QA and development communicate with each other? How are bugs tracked? How is the fix communicated back to the QA? Do you think this is the best method? Why?

The bugs that faces the client goes in DotProject which is abstracting what you call a high level bug. We promise the system would do this or have this feature and its not there and internally if I'm messing around with someone's code, and I see a software bug someway its not going to do what it is supposed to do and its not a requirement thing, its just [going to] break if you give it a negative number or divide by zero or something that goes in Bugzilla. So we've got one instance of DotProject at the customer's that's out by the firewall if it can get to, and then we've got one instance of BugZilla and that's got all the little bugs that I'm not [going to] tell a customer about.

How do you manage different customers?

I don't want one customer touching the other guys form or there's no commonality between the two. There might be, well its less work for me to put up a new instance of Apache and give them a different port and say, customer X hits port 3780 and the other one hits port 3781. That way I know their systems are separate. That would be a huge feux pas for us. It would be a major problem without financial communications. One customer wants to know what I was doing for another customer, even knowing I was doing work for another customer.

Are there any external entities that influence your process – e.g. FDA? How do they influence the project? How is this influence handled? Do they affect the design process heavily? How about the testing?

Yes, I'm once removed from those steps. I've got enough background but I can write CMM documents, 99500 spec for one, 67A for the federal government. We rarely pay somebody else to write those documents. So I would be talking to someone who specifically regards the quality specifications. Like I said, many of the customers that I had, have high risky situations type of onerous process. But generally that wouldn't result itself to somebody calling someone and saying 'did you do this, this, and this?' or ask you for a depuration that we are… where equal opportunity hirer or … I don't think that affects our core development process. They are a big part of our management but I don't think that the average developer is affected. I don't this the process changes.

How are inter team communications managed? Do you perform regular meetings? Team building?

I don't have to do that very often. On the rare cases where we have several different projects going, they are different projects for different customers. Water cooler conversation kind of takes care of trying to find the commonalities. But we don't have a formal process for that at all.