Software Development Process Models in Practical Use

Submitted in

Partial Fulfillment of the Requirements

For the ALM in IT Capstone Course

Harvard University

Extension School

May 10, 2010

Amy E. King

Table of Contents

List of Figures

1 Executive Summary

Uncertainty in business or system requirements seems an inescapable occurrence in software development projects. In 1970, when no standard process for software development existed, the Waterfall model, based on successful, established hardware development practices, was introduced to bring structure and discipline to software development.

Waterfall's rigid succession of phases requires careful, up-front planning of business requirements and technical constraints but provides no means to address change once system design and implementation have begun. Identifying inconsistencies or gaps in the system is left until Waterfall's later verification phase, at which time changes to the system's design or implementation can be costly. Waterfall is appropriate for certain types of projects such as those to develop a new system with requirements similar to a previously implemented system or a complex or expensive system requiring strict documentation and control. Waterfall's forced discipline is often cited as instructive and thus appropriate for inexperienced project managers, however, inexperience on the part of the project manager is likely to result in the very gaps in requirements which lead to Waterfall's failing. Even despite proper diligence, a project may still succumb to changes in requirements as the system is implemented or tested.

This fundamental flaw in Waterfall gave rise to agile models such as Spiral, Scrum, and Extreme Programming, each of whose central premise is that an iterative approach to software development best accommodates change. These models are widely recognized for their ability to allow a project to adapt as business and technical requirements evolve, however agile models require a shift in team and organizational culture which can be difficult to cultivate. Scrum and Extreme Programming especially rely on team communication and collaboration to move a project forward, an ability which requires small team size, good team rapport, and an enlightened organizational policy which empowers teams to self-organize and deliver product features incrementally rather than in a single, large release. If these conditions can be met, agile models can greatly boost both customer and management confidence in a project and can ensure the customer's satisfaction with the final delivered system. The Spiral model is especially applicable for projects requiring risk avoidance such as in the development of safety-critical or innovative systems. Scrum and Extreme Programming each offer the benefit of fast delivery of system features and close monitoring of customer satisfaction, making them well suited to projects whose business or technical requirements are expected to evolve during the course of the project.

When choosing a software development process model, an organization must consider its own compatibility with the model's key practices. Both an organization's culture and its policies for reporting or documentation must be weighed against the model's practices and requirements for success.

2 Introduction

The challenge of implementing a successful software development project is parodied in Figure 1 in a cartoon by Anonymous (longklaw.com, 2006). Uncertainty in business or system requirements can affect a project's success. Software development process models have evolved to address this uncertainty and increase a project's chance of successful implementation.

Having professional and academic experience as a developer on software development projects which implemented Waterfall and Extreme Programming models, the author seeks to survey four modern process models: Waterfall, Spiral, Scrum and Extreme Programming, and evaluate how each model's key practices affect its suitability for implementation in different types of software development projects. Each model has strengths and weaknesses, making it well suited for certain types of projects yet less well suited for others.

To gauge the benefits and challenges of each model in practical, real-world use, eleven students and two faculty members from the Spring 2010 Software Engineering course at the Harvard University Extension School were surveyed for their experience with and opinion of these models.

Much research exists which compares and contrasts the strengths and weaknesses of different software development process models. This paper strives to compare four common, current models and identify projects for which they are most and least well suited.

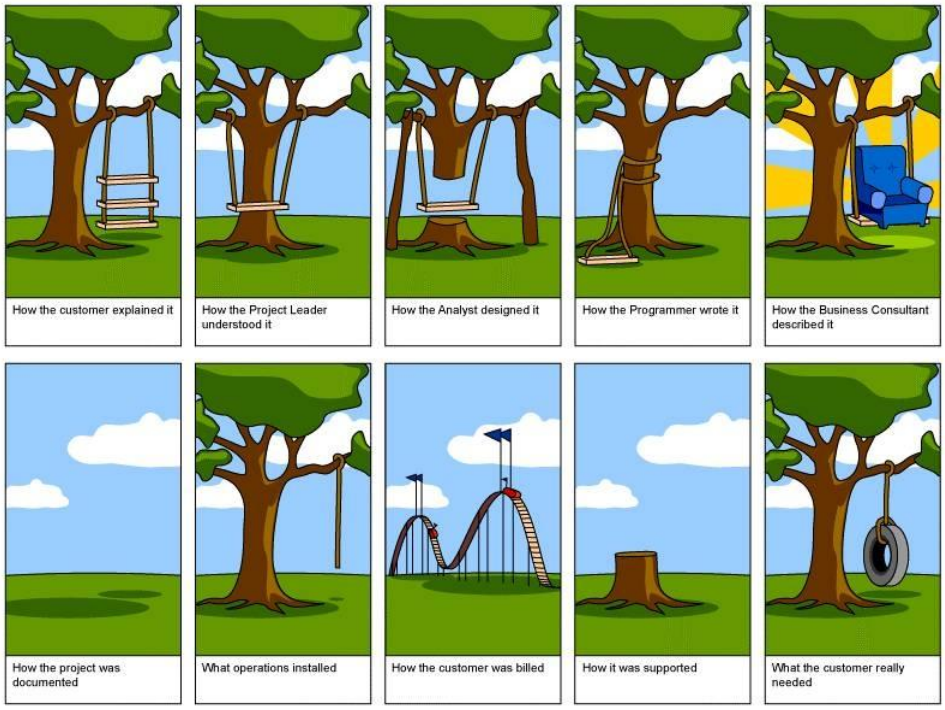

Figure 1. The software development process as parodied by Anonymous (longklaw.com, 2006)

3 The Models

Google Trends is an online tool which graphically depicts the Google search volume across time of specific search terms in relation to the total volume of Google searches (Google, 2010). It provides one method of gauging popularity or interest in a topic over time. Based on Google Trends data, four software development process models were chosen for a survey of students and faculty from the Spring 2010 Software Engineering course at the Harvard University Extension School because of their comparatively high[1] worldwide search volume between 2004 and present. Figure 2 illustrates the Google search volume of three of these models: Extreme Programming, Scrum, and Waterfall Model. As illustrated in Figure 3, Spiral has made a recent debut as a software development process model of interest.

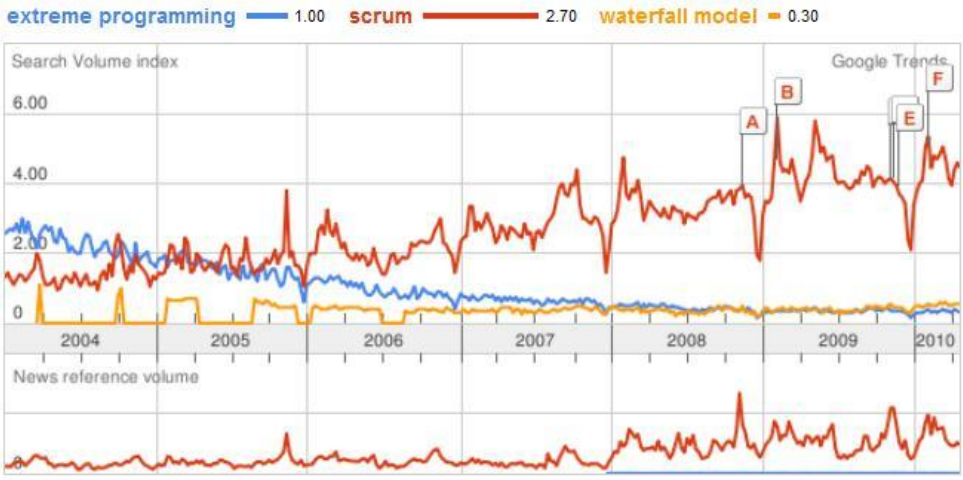

Figure 2. Google Trends data comparing Extreme Programming, Scrum, and Waterfall Model

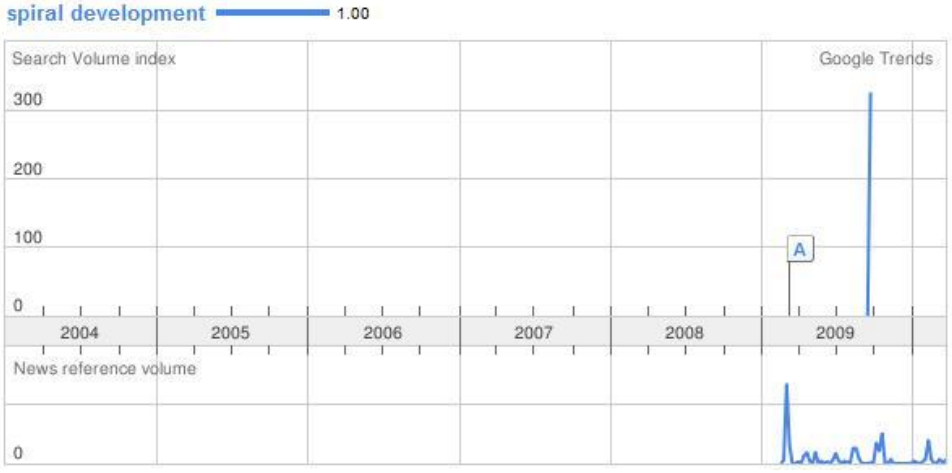

Figure 3. Google Trends data showing Spiral's recent debut as a model of interest

[1] As compared to other current software development process models such as Rational Unified Process (RUP).

2.1 Waterfall

Wikipedia credits Winston Royce for giving the first formal description of the long-standing Waterfall model in a 1970 IEEE article (Wikipedia, 2010b). Ironically, Royce was describing his personal view of a flawed process for software development that contained an inherently high risk of failure. Modeled on established hardware development processes (Sorensen, Reed, 1995), Waterfall follows a strictly sequential series of phases. Royce argues (Royce, Winston, 1970) validation of the system against customer requirements is left to the end of the process, at which time any change to the system's design or implementation is very costly:

> *The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect the development process has returned to the origin and one can expect up to a l00-percent overrun in schedule and/or costs.*

In his article, Royce makes a prophetic recommendation for adding iterative interactions between the process' sequential phases and involving the customer at several key points during the process as mechanisms for reducing both risk and cost of implementing the system.

2.1.1 Key Practices

Waterfall is usually described as having five sequential phases (Figure 4) though some descriptions of the model separate out aspects of one or more of these phases into additional, distinct phases. Each phase may begin only after its predecessor successfully concludes.

1. Requirements
   - Business and technical requirements of the system are fully defined, analyzed, and documented. This is a critical phase that informs all subsequent phases; if a business requirement or technical constraint is overlooked during this phase, the design and implementation phases can be negatively impacted, a consequence that may cascade through the remaining phases and risk project failure, schedule delay, and increased system cost (NASA, 2004).
2. Design
   - Working from the requirements document, one or more design documents are written to address system architecture, software modules and interfaces, system integration points, and user interface components (Contributor Melonfire, 2006). All business and technical requirements gathered in the requirements phase are considered and drive the design of each system component and interface.
3. Implementation
   - The system components and interfaces are coded to meet the requirements described in the design documents and are then integrated to form a working application.
4. Verification

- Verification of the system involves several aspects including unit testing of individual software components and quality assurance and use-case testing of the integrated system; acceptance testing enables the customer to interact with the system and assess its correctness (Contributor Melonfire, 2006). During the verification phase, each requirement documented in the initial requirements phase is reviewed and tested, and defects are corrected prior to deployment.

5. Maintenance
   - Following deployment, the system enters its maintenance phase, where it remains indefinitely. Incremental changes are made to the system resulting from customer requests for extended capabilities or from defects detected during productive use of the system.

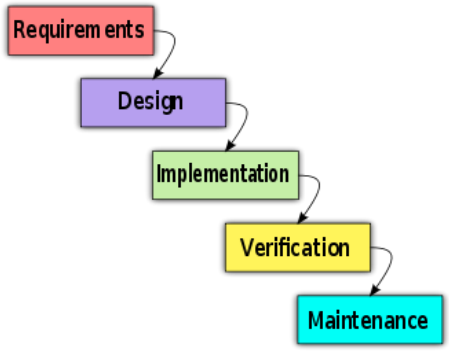

Figure 4. The Waterfall process (Wikipedia, 2009)

2.1.2 Suitability for Projects

The rigid succession of its sequential phases makes waterfall best suited for projects where the system requirements are well understood and stable at the outset of the project. Waterfall is also appropriate for projects in which a system similar to one already successfully implemented is built; reuse of the existing requirements and design makes the project easily managed by Waterfall (Sorensen, Reed, 1995).

There is disagreement on whether particularly lengthy, expensive, or complex projects benefit from Waterfall's sequential phases. As the CMS division at the U.S. Department of Health and Human Services (Centers for Medicare & Medicaid Services, 2008) determined:

> *The orderly sequence of development steps and strict controls for ensuring the adequacy of documentation and design reviews helps ensure the quality, reliability, and maintainability of the developed software.*

James Purcell however disputes this claim, asserting (Purcell, James, 2007):

> *The major weakness of the Waterfall Model is that after project requirements are gathered in the first phase, there is no formal way to make changes to the project as requirements*

> *change or more information becomes available to the project team. Because requirements almost always change during long development cycles, often the product that is implemented at the end of the process is obsolete as it goes into production... It might not [be] a good model for complex projects or projects that take more than a few months to complete.*

Waterfall is often described as supportive of inexperienced project managers who may benefit from Waterfall's forced discipline and mandate for up-front planning of business, technical, and integration requirements. However, inexperience on the part of the project manager seems likely to result in omissions in requirements which can critically unravel Waterfall's phases. Its requirement for formal business and design specifications make Waterfall suitable for projects where the system is developed by one team and transferred to a different team for life-cycle maintenance as in the case of off-site development.

Because of its rigid structure, Waterfall is poorly suited for real-time, event-driven or innovative systems (Centers for Medicare & Medicaid Services, 2008). Such projects demand flexibility to continually adapt to changes in business, design, or technical requirements, a trait incompatible with Waterfall's structured phases. Further, its documentation overhead makes Waterfall unsuitable for projects requiring fast delivery of a system. Waterfall should not be attempted for projects whose business or technical requirements are not thoroughly understood at the project's start.

2.2 Spiral

Originally described by Barry Boehm in 1988, the Spiral model of software development focuses on minimizing risk to a project. In a later article, Boehm and Hansen (Boehm, Barry and Hansen, Wilfred, 2001) summarize Spiral's goals:

> *The spiral development model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software-intensive systems. It has two main distinguishing features. One is a cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.*

The model's main features called out in this description each effect a reduction in risk. Spiral's cyclic nature continually identifies and addresses risk, and the customer's active involvement ensures project deliverables remain compatible with the customer's goals.

2.2.1 Key Practices

The Spiral model iterates through four quadrants of key practices, as illustrated in Figure 5. Each iteration develops a version of the system more refined and expansive than the previous version until the system meets all customer requirements and no longer requires further enhancement.

1. Determine Key Components
   - Prior to entering the spiral, key components are concurrently rather than sequentially identified. These include hardware and technological requirements, system architecture and design, and integration requirements. By identifying these components concurrently and adjusting decisions to remain compatible with other key components, the project is not unduly constrained by prior, immutable decisions.
2. Identify Objectives, Constraints, and Alternatives
   - Each loop through the spiral begins with identifying the objectives of the iteration, among them the customer's highest priority goals. Constraints and dependencies among the objectives are assessed, and alternative approaches and risks are identified.
3. Evaluate Risks, Constraints, and Alternatives
   - Risks to the project introduced by the objectives identified in the previous step are evaluated. Technological, integration, or other constraints are assessed and alternative approaches are weighed. Components and approaches which present the least risk to the project are chosen for implementation.
4. Develop and Test the System
   - Using decisions made during evaluation of the project's objectives and key components, detailed requirements of each component are flushed out and the components are developed and tested. Testing includes unit testing of individual components as well as integration and acceptance testing.
5. Assess the Delivered System and Plan the Next Iteration
   - In the final quadrant of the spiral, the customer evaluates the end-product delivered during the iteration and determines the features and objectives that are still needed. These requirements are used as input into the next loop through the spiral.

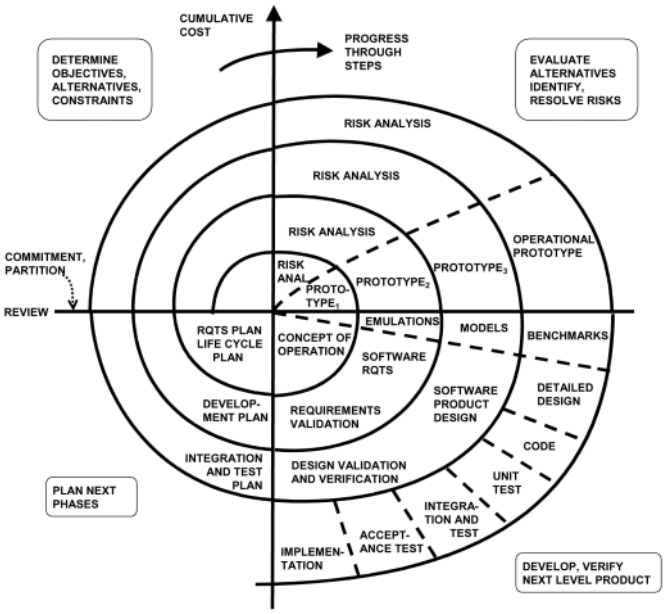

Figure 5. The Spiral process (Boehm, Barry and Hansen, Wilfred, 2001)

Three anchor point milestones spaced throughout the spiral ensure the customer's continued satisfaction with and commitment to the project. A Life Cycle Objective (LCO) milestone during determination of the project's key components ensures the initial vision of the system meets the customer's business requirements. Prior to entering iterative development, a Life Cycle Architecture (LCA) milestone requires the customer to support initial deployment of the system. Finally, prior to release of the first version of the system, an Initial Operating Capability (IOC) milestone commits the customer to supporting system operations after release (Boehm, Barry and Hansen, Wilfred, 2001). Together, the milestones confirm the system under development satisfies the customer's business requirements and establishes the customer's continued investment in the project.

2.2.2 Suitability for Projects

Spiral's iterative approach makes it a beneficial model for projects requiring risk avoidance. Projects to develop safety-critical or real-time systems (Centers for Medicare & Medicaid Services, 2008) or computation-heavy decision-support systems (DeGrace, Peter and Stahl, Leslie, 1990) require the ability to continuously assess the accuracy and direction of the project, and Spiral's short iterations enable close monitoring of a project's objectives, constraints, and risks. Adam Kolawa notes Spiral's additional suitability for innovative projects (Kolawa, Adam, 2010):

> *The spiral process works best if you have a ground-breaking project of undefined scope and plan on continually redefining and perfecting particular features throughout the course of the project... [S]hort iterations allow the team to quickly show the customer the results of the latest request. This allows for rapid feedback on the success of the most recent iteration*

> *and the direction of the next one. Such frequent and rapid feedback is not necessary for traditional projects, but is the lifeblood of innovative ones.*

Because each iteration presents a new opportunity to identify and evaluate objectives, Spiral is also suitable for projects whose business requirements are not fully known at the start of the project. This benefit however comes at a cost—with the final assessment of one iteration feeding new objectives into the next, Spiral projects lack a clear end date (Centers for Medicare & Medicaid Services, 2008); development or project management resources may continue to be occupied by a project for longer than necessary. Spiral is therefore not suitable for projects needing to conserve human resources.

The Spiral model offers the flexibility to incorporate other software development process models into an iteration depending on the iteration's particular risks (Centers for Medicare & Medicaid Services, 2008). This flexibility makes Spiral suitable for extended or complex projects where each iteration may require a different approach than the last but also requires an experienced project manager to maintain order and control throughout the project's evolution.

2.3 Scrum

With a focus on project management, Scrum seeks to increase the predictability of project success and reduce risk (Schwaber, K., and Sutherland, J., 2010) by providing a framework for an iterative, incremental approach to software development (Figure 6).

Scrum was formally presented to the world at the 1995 OOPSLA conference by Jeff Sutherland, Ken Schwaber, and Mike Beedle. Before that time, Scrum had been slowly gaining definition and attention since its first mention as a product development model by DeGrace and Stahl (DeGrace, Peter and Stahl, Leslie, 1990) five years earlier. They were the first to associate the word *scrum* with product development after-- according to Wikipedia (Wikipedia, 2010a)-- an analogy made by Takeuchi and Nonaka (Takeuchi, H. and Nonaka, I., 1986) to rugby's scrum formation in which all team members cooperate to achieve a common goal. The first software development project to employ Scrum was undertaken by Jeff Sutherland, John Scumniotales, and Jeff McKenna at the Easel Corporation in 1993 (Sutherland, Jeff, 2004).

Scrum identifies individuals as pigs and chickens according to their level of involvement in a project. Pigs, e.g., Scrum team members, are committed to the project and incur more personal risk than chickens, e.g., managers and end-users, who are only marginally involved in the project. According to Ken Schwaber (Schwaber, K. and Sutherland, J., 2010), The terms originate from the story:

> *A chicken and a pig are together when the chicken says, "Let's start a restaurant!" The pig thinks it over and says, "What would we call this restaurant?" The chicken says, "Ham n' Eggs!" The pig says, "No thanks, I'd be committed, but you'd only be involved!"*

2.3.1 Key Practices

Scrum is a framework consisting of three components.

1. The Scrum Team
    - Scrum teams are self-organizing, cross-functional, and collaborative teams of 7±2 people (Schwaber, K. and Sutherland, J., 2010). Team members are chosen to ensure the team possesses all skills needed to achieve project goals.
    - Each team member plays one of three roles: the ScrumMaster who guides the team and resolves obstacles; the Product Owner who is responsible for identifying and prioritizing business requirements; or the Team Member who is responsible along with other team members for development, quality assurance, business analysis, and system design.
2. Ceremonies, a.k.a., Time-Boxes
    - Referred to as ceremonies by the Scrum Alliance (Scrum Alliance, 2009a) and time-boxes by Schwaber and Sutherland (Schwaber, K. and Sutherland, J., 2010), Scrum employs several planning and review meetings to either create a plan of action at the beginning of a project phase or to review the progress and productivity at the end of a phase.
    - Scrum's release planning meeting establishes the set of Product Owner-written user stories that will be deployed in a single release.
    - The release is then divided into development sprints, each between two to four weeks in duration (Scrum Alliance, 2009b), which schedule specific user stories for development and attempt to deliver a potentially production-ready set of features. At the outset of each sprint, a sprint planning meeting prioritizes the user stories to be developed during the sprint.
    - Brief, daily Scrum meetings enable team members to discuss progress, status, and to identify obstacles which the ScrumMaster or other team members may assist in removing.
    - At the conclusion of each sprint, a sprint review meeting is held to demonstrate to the Product Owner the current state of the system and to review and reprioritize outstanding user stories for the next sprint.
3. Artifacts
    - The product backlog is most often comprised of user stories, though use cases are also seen (Schwaber, K. and Sutherland, J., 2010). It represents the set of outstanding system requirements, new features, and enhancements that will be developed during the current release.
    - The sprint backlog, developed during each sprint planning meeting, is the set of system requirements, new features, and enhancements that will be developed during the current sprint. While the product backlog items included in a sprint are fixed at the start of each sprint, the sprint backlog expands and contracts as new tasks are added to implement user stories or other tasks are eliminated from scope.

- A burndown chart, maintained either separately for the release and sprint backlog or as a single burndown chart encompassing both, tracks the rate of progress of development through the product or sprint backlog. It is used to measure the Scrum team's development capacity during a sprint and compare that capacity to the volume of remaining work.

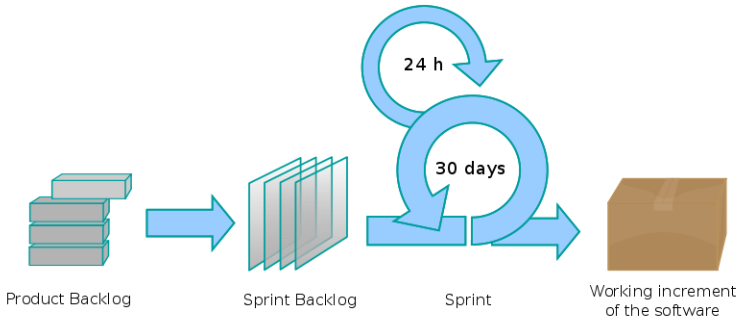

Figure 6. The Scrum process (Wikipedia, 2008)

2.3.2 Suitability for Projects

Like other agile models, Scrum is best suited for projects employing small teams because of its reliance on cross-functional collaboration within the team. Its self-organizing approach also requires the Scrum team to be dedicated to a single project (Santhosh, Srinivasan, 2002); having team members simultaneously allocated to other projects can hinder the process of self-organizing when team members are diverted to other projects. Scrum additionally requires a willingness on the part of management to trust the Scrum team and allow it to self-organize as well as a willingness on the part of the customer (as Product Owner) to be an active participant at Scrum meetings.

Each of Scrum's limitations and benefits derive from its model of cross-functional, self-organizing teams. The model promotes high productivity with fast delivery of an incrementally more complex system, and this can demonstrate to a customer the project's continuous progress. As Ken Schwaber (Net Solutions, 2008) notes however, for Scrum to be successful within an organization, an often difficult cultural change is needed:

> *Several changes, or cultural shifts, are required to use Scrum. The first is to forget predictive, waterfall thinking. The second is to realize that self-management is a much better practice for productivity and creativity. The third is to understand that cross-functional teams produce more robust products. All of these changes are extremely difficult, regardless of the country. I'd say that the United States has a lot of trouble going from top-down, command and control to self-management; we believe that the only way to get something done is to make it get done.*

Scrum is widely recognized as being well suited for organic projects whose business requirements are not fully formed at the start of the project. It is best suited for the development of new systems rather than enhancement of existing systems (Santhosh, Srinivasan, 2002). Its de-emphasis on documentation makes it unsuitable for organizations requiring formal documentation or for projects where the system is developed by one team and transferred to a different team for life-cycle maintenance as in the case of off-site development (Sumedh, 2007).

Because of its reliance on trust and collaboration, Scrum is more successful when both the Scrum team and management are experienced in agile practices (Chou, Norman, 2005). Though Scrum rejects formal oversight from management, daily Scrum and sprint review meetings offer sufficient transparency into the project's status and progress to both customers and management.

2.4 Extreme Programming

First seen in 1996, Extreme Programming is an agile process that emphasizes customer satisfaction in its approach to projects. Don Wells (Wells, J. Donovan, 2009a) claims Extreme Programming improves software development projects in five ways: communication, simplicity, feedback, respect, and courage:

> *Programmers constantly communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested. Every small success deepens their respect for the unique contributions of each and every team member. With this foundation Extreme Programmers are able to courageously respond to changing requirements and technology.*

Extreme Programming's additional focus on the satisfaction of project team members is apparent. By eliminating points of frustration and emphasizing satisfaction of both customers and project team members, Extreme Programming endeavors to foster an environment where collaboration, empowerment, and contentment are sustainable benefits that lead ultimately to higher productivity.

2.4.1 Key Practices

Extreme Programming's key practices, illustrated in Figure 7, fall into five categories.

1. Planning
    - A project begins with user stories written by the customer. Each user story describes a business scenario and is used both to estimate development effort which informs release planning and to drive later acceptance testing.

- Release planning then takes a broad view of the project's objectives and ideally sets a goal of deploying 80±20 (Wells, J. Donovan, 2009b) user stories to the customer in a single release.
- The release is then divided into development iterations, each between one to three weeks in duration, which schedule specific user stories for development.

2. Managing
   - Managing the project comprises providing an environment that encourages collaboration, communication, and cross-training among project team members. An open workspace and daily "stand-up" meetings ensure regular communication among project team members and enables all team members to have familiarity with all parts of the system.
   - Project velocity, though something of a misnomer, is a measure of the project team's development capacity during an iteration. It is described by Don Wells (Wells, J. Donovan, 1999) as the sum of development time estimates for user stories completed during an iteration and not as a ratio of that sum against total user stories, project team size, duration of the iteration, or any other factor.
3. Designing
   - System design begins with simplicity. Choosing a design that is easy to explain to new team members and easily understood and tested ensures new members can begin to contribute to the project quickly, tests may be automated, and defects are quickly recognized and fixed. Design simplicity is reinforced by not adding features to the system before they are required by a user story.
   - Extreme Programming employs Class, Responsibilities, and Collaboration (CRC) cards as a tool for design planning. A set of CRC cards is analogous to an abbreviated UML diagram; each card represents an object in the system, and its primary responsibilities and collaborating objects are noted.
   - During development, spike solutions and code refactoring serve as opportunities to redesign and further simply the system. Spike solutions allow a particularly challenging or high-risk user story to be prototyped outside of the system in order to identify the best possible solution.
4. Coding
   - Extreme Programming encourages developers to work in all parts of the code base to prevent bottle-necks and knowledge islands that can reduce a project's velocity, a concept Extreme Programming calls collective ownership. Adhering to coding standards is an important practice that simplifies collective ownership and can reduce code maintenance costs.
   - Pair programming is Extreme Programming’s most counterintuitive practice. By having two developers writing code collaboratively, side-by-side at a single computer, Don Wells (Wells, J. Donovan, 1997) claims quality of the system code base is greatly increased with no cost to productivity. This higher quality translates to higher efficiency and lower maintenance costs of the delivered system. Kent Beck (Beck, Kent, 2000) goes further to suggest Pair Programming reinforces collective ownership and design

simplicity by subjecting all code to instant peer-review which encourages conscientious coding practices. To be effective however, Pair Programming requires honesty, patience, and humility between the paired developers, which can be difficult to cultivate.

5. Testing
   - Another counterintuitive practice is Extreme Programming's insistence that unit tests be written before the functionality they test. Though Kent Beck (Beck, Kent, 2000) allows that a test may be written before its functionality only under particular circumstances of uncertainty or complexity, Don Wells (Wells, J. Donovan, 2000a) proposes writing the unit test first has several benefits including making subsequent coding of the functionality easier and faster, reinforcing simplicity of system design, and helping to flush out misunderstandings in the requirements of a user story.
   - Acceptance tests are business scenarios that demonstrate a user story has been correctly implemented. Like user stories, they are specified by the customer, and it is the customer's responsibility to verify an acceptance test's validity.

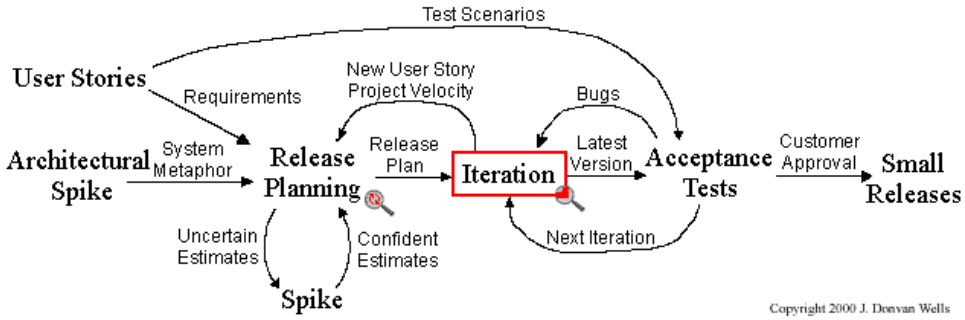

Figure 7. The Extreme Programming process (Wells, J. Donovan, 2000b)

2.4.2 Suitability for Projects

Extreme Programming has a number of characteristics that limit its suitability for projects. Its reliance on team communication and collaboration to move the project forward make the process non-trivial for teams in which team members are inexperienced with Extreme Programming's practices or in which team members simply don't know each other well (Nawrocki, J. R. et al., 2002). Constant involvement of the customer may present a challenge to the customer's schedule which can impact project velocity when the team must wait for customer feedback or approvals. Additionally, an organization whose process requires strict documentation of a system's requirements, architecture, or deliverables is in conflict both with Extreme Programming's agile approach and with its de-emphasis on formal documentation. Kent Beck (Beck, Kent, 2000) acknowledges additional limitations:

> *The exact limits of XP aren't clear yet. But there are some absolute showstoppers that prevent XP from working—big teams, distrustful customers, technology that doesn't support graceful change.*

Extreme Programming is most well suited for small to medium-sized projects employing small teams. Small team size promotes the communication and collaboration needed within Extreme Programming teams, and development of smaller systems can be more flexible and require less formal documentation. Large or complex systems tend to have an organizational requirement for formal documentation as a mechanism to constrain system implementation and maintenance costs. Even a system that was initially successfully implemented using Extreme Programming may grow, over time, too large to be further enhanced using Extreme Programming practices (Paulk, Mark, 2001).

Extreme Programming is also very well suited for projects in which business requirements are not fully known or understood at the start of the project. Its agile, iterative approach and focus on customer involvement allow system development to move forward while specific functionality and features evolve incrementally as the project progresses. While this organic approach can increase risk for higher maintenance costs because of a lack of design or architecture planning (Paulk, Mark, 2001), Extreme Programming's practices of simplicity and refactoring can mitigate this risk.

3 Survey Findings

In a survey of eleven students and two faculty members from the Spring 2010 Software Engineering course at the Harvard University Extension School, seven respondents reported having used one or more of Waterfall, Spiral, Scrum, or Extreme Programming software development process models in an academic or professional setting (Figure 8).

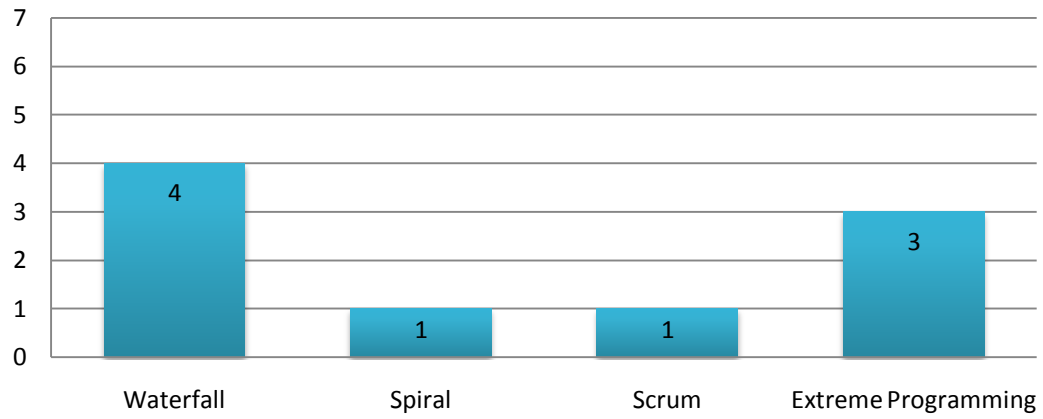

Figure 8. Survey respondent experience

Four respondents reported experience with the Waterfall model. They reported the projects implemented using Waterfall included several characteristics of a suitable Waterfall project including:

- An organizational policy requiring strict approval of project phases or deliverables

- A project or organizational requirement for formal system documentation
- The customer having thorough understanding of business needs
- An inexperienced project manager

Some respondents reported that projects benefited from Waterfall's structure, which provided a well-defined timeline that aided in project planning. Others indicated that process overhead and an organizational reluctance to enforce the model's practices undermined Waterfall's potential benefits.

Despite 75% of respondents reporting the customer had the requisite understanding of business needs and 100% of respondents reporting having implemented Waterfall's requirements and design phases, 100% of those using Waterfall reported that system or business requirements remained in flux throughout the project. 100% of respondents also reported one or more of Waterfall's phases were problematic. It is possible this is a result of project manager inexperience or of the organization's failure to enforce rigorous approvals at each phase, however, it also may demonstrate the consequence of an unrealistic model whose key practices assume stable, unchanging requirements.

One respondent reported experience with the Spiral model, and reported the projects implemented using Spiral included many characteristics of a suitable Spiral project including:

- A project having risk-avoidance as a high priority
- Development of a large or complex system
- Development of a safety-critical system
- Business or system requirements not fully known at the start of the project

The respondent also reported however an inexperienced project manager and an organizational policy for project reporting which assumed the use of Waterfall's phased practices. As a result, the respondent indicated all of Spiral's key practices remained somewhat problematic for about one year while both management and developers adapted to the Spiral model. Such growing pains during transition between models is not unexpected. After the transition, the respondent reported the team was able to effectively apply Spiral's practices to new projects.

One respondent also reported experience with the Scrum model, and reported the projects implemented using Scrum included many characteristics of a suitable Scrum project including:

- The customer being highly available and involved in the project
- Experienced developers with good rapport
- Development of a new product
- A requirement for fast delivery of features
- Business or system requirements not fully known at the start of the project

The respondent reported that many of Scrum's key practices were of great benefit to the project including short iterations that each delivered some system features, customer involvement, guidance of the ScrumMaster, and the use of user stories to drive development. Daily Scrum meetings and the management of the product backlog however presented challenges. These challenges may have been due to project team members being simultaneously allocated to multiple projects or a project manager inexperienced in Scrum.

Three respondents reported experience with Extreme Programming. They reported the projects implemented using Extreme Programming included several characteristics of a suitable Extreme Programming project including:

- The customer being highly available and involved in the project
- A small to medium-sized project
- Experienced developers with good rapport
- Experienced project manager
- Business or system requirements not fully known at the start of the project

All respondents reported that projects benefited from many of Extreme Programming's key practices. 100% reported short iterations that each delivered some system features, the use of user stories to drive development, and release planning to be of greatest benefit. The customer's close involvement in the project, the use of spike solutions, regular refactoring, and team "stand-up" meetings were also noted by all respondents as beneficial.

There was disagreement on the efficacy of Extreme Programming's more unusual practices of pair programming and writing unit tests before functionality. Of the three respondents, one indicated both practices were of great benefit to the project, another indicated these practices were problematic, and the third respondent indicated the practices were not used because of resistance and discomfort on the part of developers.

4 Summary and Conclusions

In the absence of a process model for software development, the early Waterfall model was patterned after established, successful hardware development practices. Agile models like Spiral, Scrum, and Extreme Programming followed in response to Waterfall's inability to accommodate changing business or system requirements, a seemingly inescapable occurrence in software development projects.

Each model however has its place in modern software development. The Waterfall model is appropriate for the development of a new system whose requirements are similar to a previously implemented system. It offers stability and control in the development of particularly complex or expensive system components. Because it provides no mechanism to address changes in business or system requirements however, it is only suitable for projects whose

requirements are guaranteed not to change during project implementation, an often unrealistic constraint. Waterfall also may be unsuitable for inexperienced project managers likely to omit such requirements during Waterfall's requirements phase. The Spiral model is well suited for innovative or safety-critical systems where risk to the project must be continually monitored and quashed. It also provides the ability to identify during each iteration new business or system requirements, making it well suited to complex projects or projects where such requirements are expected to evolve. Spiral projects can however be resource intensive, consuming customer, project manager, and developer time for longer periods than models with well defined deadlines. Like Spiral, Scrum and Extreme Programming are both very well suited to projects whose business or system requirements are expected to emerge during the course of the project. The practice of iterative development makes both very effective at fast delivery of an incrementally more feature-rich system, an ability which can greatly boost both customer and management confidence in a project. Their reliance on team autonomy and active customer involvement however can require a difficult cultural shift within an organization.

Agile models are responsive to the inevitable changes in requirements that occur in software development projects, but it is not only their iterative approach which affords such responsiveness. Their ability to accommodate change requires small project teams to limit the overhead associated with communication and collaboration. Their ability to be responsive requires an organizational culture that promotes team empowerment and early delivery of results over structured phases and rigid documentation. An organization instituting an agile model must also establish control mechanisms to prioritize the "scope creep" which can often occur in agile models, a result of each iteration providing the customer an opportunity to identify new possibilities for the system.

In adopting a software development process model, the culture of the organization plays a significant role. Organizations requiring detailed business or system documentation and formal approvals between a project's phases will find agile models flimsy. Organizations wishing to offer fast delivery of results to customers or implement innovative systems will find Waterfall's structure stifling. The flexibility of agile models can offer a compromise. User stories in Scrum or Extreme Programming can describe a need for documentation. Spiral's ability to incorporate other process models into its iterations can address localized requirements for more structure. The customer's active involvement in each of these agile models can serve to secure necessary approvals and commitments. In choosing to embrace a software development process model, an organization must consider not only the model's strengths and weaknesses but the organization's own compatibility with the model's practices and requirements for success..

5 Glossary

Agile Process

An adaptive process where requirements for and refinements to a system evolve over multiple iterations and through collaboration among project team members.

IEEE

Institute of Electrical & Electronics Engineers, a professional organization dedicated to the advancement of technological innovation and excellence.

OOPSLA

Object-Oriented Programming, Systems, Languages & Applications conference, an annual conference hosted by the Association for Computing Machinery (ACM).

UML

Unified Modeling Language, a standardized notation for graphically modeling a software system's components.

6 References

Beck, Kent (2000). Extreme Programming Explained: Embrace Change, Addison-Wesley: pp. 58, 69, 89.

Boehm, Barry and Hansen, Wilfred (2001). The Spiral Model as a Tool for Evolutionary Acquisition, CrossTalk: The Journal of Defense Software Engineering, May 2001. http://www.stsc.hill.af.mil/Crosstalk/2001/05/boehm.html, retrieved April 2010.

Centers for Medicare & Medicaid Services (2008). Selecting a Development Approach. http://www.cms.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf, retrieved April 2010.

Chou, Norman (2005). Comparing and Contrasting Agile Methods. http://csse.usc.edu/classes/cs577b_2005/presentation/Norman.ppt: p. 5, retrieved May 2010.

Contributor Melonfire (2006). Understanding the pros and cons of the Waterfall Model of software development. http://articles.techrepublic.com.com/5100-10878_11-6118423.html, retrieved April 2010.

DeGrace, Peter and Stahl, Leslie (1990). Wicked Problems, Righteous Solutions: A Catalogue of Modern Engineering Paradigms. Prentice Hall: pp. 116-117.

Google (2010). About Google Trends. http://www.google.com/intl/en/trends/about.html, retrieved April 2010.

Green, Darryl and DiCaterino, Ann (1998). A Survey of System Development Process Models, Center for Technology in Government, University at Albany, SUNY: pp. 5. http://www.ctg.albany.edu/publications/reports/survey_of_sysdev/survey_of_sysdev.pdf. retrieved May 2010.

Kolawa, Adam (2010). Which Development Method Is Right for Your Project? http://www.stickyminds.com/r.asp?F=DART_3152, retrieved May 2010.

longklaw.com (2006). Software Development Process. http://www.longklaw.com/2006/07/software-development-process, retrieved April 2010.

NASA (2004). The Standard Waterfall Model for Systems Development. http://web.archive.org/web/20050310133243/http://asd-www.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm, retrieved April 2010.

Nawrocki, J. R., Bartosz, W., and Wojciechowski, A. (2002). Comparison of CMM Level 2 and eXtreme Programming, Proceedings of the 7th International Conference on Software Quality: p. 296. http://info.iet.unipi.it/~vaglini/GQ/Articoli/CMMIeXtreme.pdf, retrieved April 2010.

Net Solutions (2008). Interview with Ken Schwaber. http://www.agilecollab.com/interview-with-ken-schwaber, retrieved May 2010.

Paulk, Mark (2001). Extreme Programming from a CMM Perspective, IEEE Software, November/December 2001: pp. 7-8. ftp://ftp.sei.cmu.edu/pub/documents/articles/pdf/xp-from-a-cmm-perspective.pdf, retrieved May 2010.

Paulk, M., Weber, C., Garcia-Miller, S., et al. (1993). Key Practices of the Capability Maturity Model[SM], Version 1.1: pp. O-8, O-14-O-19, http://www.sei.cmu.edu/library/abstracts/reports/93tr025.cfm, retrieved April 2010.

Purcell, James (2007). Comparison of Software Development Lifecycle Methodologies. http://www2.giac.org/resources/whitepaper/application/217.php, retrieved May 2010.

Royce, Winston (1970). Managing the Development of Large Software Systems, Proceedings of IEEE WESCON 26 (August): pp. 329-330, http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf, retrieved April 2010.

Schwaber, K. and Sutherland, J. (2010). Scrum Guide. http://www.scrum.org/storage/scrumguides/Scrum Guide.pdf: pp. 3, 6, 8-9, 16, retrieved April 2010.

Scrum Alliance (2009a). Scrum Ceremonies. http://www.scrumalliance.org/pages/scrum_ceremonies, retrieved Aril 2010.

Scrum Alliance (2009b). What is Scrum? http://www.scrumalliance.org/learn_about_scrum, retrieved Aril 2010.

Sorensen, Reed (1995). A Comparison of Software Development Methodologies, CrossTalk: The Journal of Defense Software Engineering, January 1995. http://www.stsc.hill.af.mil/crosstalk/1995/01/comparis.asp, retrieved May 2010.

Srinivasan, Santhosh (2002). Scrum, http://courses.cs.tamu.edu/lively/cpsc606/Scrum.ppt, retrieved May 2010.

Sumedh (2007). Scrum Pros and Cons. http://sumedhsays.blogspot.com/2007/07/scrum-pros-and-cons.html, retrieved May 2010.

Sutherland, Jeff (2004). Agile Development: Lessons Learned From The First Scrum. http://www.scrumalliance.org/resources/35: p. 1, retrieved April 2010.

Takeuchi, H. and Nonaka, I. (1986). The New New Product Development Game. Harvard Business Review. http://apln-richmond.pbwiki.com/f/New%20New%20Prod%20Devel%20Game.pdf, last retrieved 2008.

Wells, J. Donovan (1997), Pair Programming, http://www.extremeprogramming.org/rules/pair.html, retrieved April 2010.

Wells, J. Donovan (1999), Project Velocity, http://www.extremeprogramming.org/rules/velocity.html, retrieved April 2010.

Wells, J. Donovan (2000a). Code the Unit Test First, http://www.extremeprogramming.org/rules/testfirst.html, retrieved April 2010.

Wells, J. Donovan (2000b). Extreme Programming Project, http://www.extremeprogramming.org/map/project.html, retrieved April 2010.

Wells, J. Donovan (2009a). Extreme Programming: A gentle introduction, http://www.extremeprogramming.org, retrieved April 2010.

Wells, J. Donovan (2009b). User Stories, http://www.extremeprogramming.org/rules/userstories.html, retrieved April 2010.

Wikipedia (2008). File:scrum process.svg. http://en.wikipedia.org/wiki/File:Scrum_process.svg, retrieved April 2010.

Wikipedia (2009). File:Waterfall model.svg. http://en.wikipedia.org/wiki/File:Waterfall_model.svg, retrieved April 2010.

Wikipedia (2010a). Scrum (development). http://en.wikipedia.org/wiki/Scrum_(development), retrieved April 2010.

Wikipedia (2010b). Waterfall model. http://en.wikipedia.org/wiki/Waterfall_model, retrieved April 2010.