2007

XP PROJECT MANAGEMENT

Craig William Macholz
The University of Montana

XP PROJECT MANAGEMENT

By

Craig William Macholz

BS in Business Administration, The University of Montana, Missoula, MT, 1997

Thesis

presented in partial fulfillment of the requirements

for the degree of

Master of Science

in Computer Science

The University of Montana

Missoula, MT

Autumn 2007

Macholz, Craig, M.S., December 2007 Computer Science

Extreme Programming Project Management

Chairperson: Dr. Joel Henry

Extreme programming project management examines software development theory, the extreme programming process, and the essentials of standard project management as applied to software projects. The goal of this thesis is to integrate standard software project management practices, where possible, into the extreme programming process. Thus creating a management framework for extreme programming project management that gives the extreme programming managers the management activities and tools to utilize the extreme programming process within a wider range of commercial computing organizations, relationships, and development projects. The author speculates that the lack of organizational adoption and mainstream integration of the extreme programming process is connected to the lack of standard project management structure that allows for greater control of scheduling, cost, and quality. This paper will create a basic framework that will be used to improve the extreme programming process and identify its shortcomings.

ACKNOWLEGEMENTS

Thank you to Joel Henry, Ph.D., Shawn Clouse, Ph.D., and Yolanda Reimer, Ph.D for their valued input and support. Special thanks to my wife Lee, daughter Kaylee and my parents, Jim and Sue Macholz.

TABLE OF CONTENTS

LIST OF TABLES VII

LIST OF ILLUSTRATIONS VII

CHAPTER 1 – INTRODUCTION 1

1.1– Problem Statement 1
1.2 – Approach/Methods 2
1.3 – Software Process Background 3

CHAPTER 2 – OVERVIEW OF EXTREME PROGRAMMING 7

2.1 – The Purpose of XP 8
2.2 – Rights Within XP 9
2.3 – XP Core Values 11
2.4 – XP Core Practices 12
2.5 – The Planning Game 13
2.5.1 – User Stories 13
2.5.2 – Story Estimation 16
2.5.3 – Release Planning 18
2.5.4 – Iteration Planning 20
2.6 – Small Releases 21
2.7 – System Metaphor 21
2.8 – Simple Design 22
2.9 – Testing 23
2.9.1 – Unit Tests 23
2.9.2 – Acceptance Tests 25
2.10 – Refactoring 26
2.11 – Pair Programming 27
2.12 – Collective Ownership 28
2.13 – Continuous Integration 29
2.13.1 – Releasing Changes 29
2.13.2 – Possible Problems and Fixes 30
2.13.3 – Slow Merges 30
2.13.4 – Lost Changes 31
2.14 – 40-Hour Workweek 31
2.15 – On-Site Client 31
2.16 – Coding Standards 32
2.17 – XP Practices Summary 33
2.18 – Roles of the XP Team Members 34
2.18.1 – Programmer 34
2.18.2 – Customer 35
2.18.3 – Tester 36
2.18.4 – Tracker 36
2.18.5 – Coach 36
2.18.6 – Consultant 37
2.18.7 – Manager 37
2.19 – XP Management Strategy 39
2.19.1 – Metrics 40

2.19.2 – Coaching *42*
2.19.3 – Intervention *42*
2.20 – FACILITIES 43
2.21 – STEERING 44
2.21.1 – Iteration Steering *44*
2.21.2 – Release Steering *46*

CHAPTER 3 – OVERVIEW OF SOFTWARE PROJECT MANAGEMENT 48

3.1 – INTRODUCTION TO THE BASICS OF SOFTWARE PROJECT MANAGEMENT 48
3.1.1 – Estimation & Planning Overview *49*
3.1.2 – Monitoring Overview *49*
3.1.3 – Controlling & Adjusting Overview *50*
3.1.4 – Completion Overview *51*
3.2 – ESTIMATE & PLAN 52
3.2.1 – Staff—Estimate & Plan *52*
3.2.2 – Software Development Process—Estimate & Plan *55*
3.2.3 – Risk Assessment—Estimate & Plan *58*
3.2.4 – Resources—Estimate & Plan *59*
3.2.5 - Effort—Estimate & Plan *62*
3.2.6 – Schedule—Estimate & Plan *63*
3.2.7 – Measurements—Estimate & Plan *65*
3.2.8 – Deployment—Estimate & Plan *67*
3.3 – MONITOR 68
3.3.1 – Staff—Monitor *69*
3.3.2 – Software Development Process—Monitor *70*
3.3.3 – Risk Assessment—Monitor *70*
3.3.4 – Resources—Monitor *71*
3.3.5 – Effort—Monitor *72*
3.3.6 – Schedule—Monitor *73*
3.3.7 – Measurements—Monitor *75*
3.3.8 – Deployment—Monitor *76*
3.4 - CONTROL & ADJUST 77
3.4.1 – Staff—Control & Adjust *77*
3.4.2 – Software Development Process—Control & Adjust *79*
3.4.3 – Risk Assessment—Control & Adjust *79*
3.4.4 – Resources—Control & Adjust *80*
3.4.5 – Effort—Control & Adjust *81*
3.4.6 – Schedule—Control & Adjust *82*
3.4.7 – Measurements—Control & Adjust *83*
3.4.8 – Deployment—Control & Adjust *84*
3.5 – COMPLETION 85
3.5.1 – Staff—Completion *85*
3.5.2 – Software Development Process—Completion *88*
3.5.3 – Risk Assessment—Completion *89*
3.5.4 – Resources—Completion *89*
3.5.5 – Schedule—Completion *91*
3.5.6 – Effort—Completion *92*
3.5.7 – Measurements—Completion *93*
3.5.8 – Deployment—Completion *94*

CHAPTER 4 – MAPPING SOFTWARE PROJECT MANAGEMENT TO XP 95

4.1 – INTRODUCTION 95
4.2 – XP PRACTICES 95

4.2.1 – The Planning Game *96*
4.2.2 – Small Releases *113*
4.2.3 – System Metaphor *120*
4.2.4 – Simple Design *124*
4.2.5 – Testing *129*
4.2.6 – Refactoring *137*
4.2.7 – Pair Programming *143*
4.2.8 – Collective Ownership *151*
4.2.9 – Continuous Integration *154*
4.2.10 – 40-Hour Workweek *160*
4.2.11 – On-Site Client *162*
4.2.12 – Coding Standards *171*

CHAPTER 5 – CONCLUSION 175

BIBLIOGRAPHY 179

APPENDIX A – SOFTWARE PROJECT MANAGEMENT TASKS MAPPED TO XP PRACTICES 180

LIST OF TABLES

Table Page
1. Software development processes. 56
2. Four steps for rescheduling smartly. 83

LIST OF ILLUSTRATIONS

Figure Page
1. The waterfall model. 4
2. The spiral model. 4
3. A comparison of software design models. 6
4. Cost of process changes. 8
5. The Planning Game. 15
6. Sample user stories. 16
7. Story estimation. 17
8. Interconnectivity of the 12 core XP practices. 33
9. Tracking acceptance tests. 41
10. Equation for estimating staff size. 52
11. Alternative paths for estimation and planning. 68
12. An overview of the planning game and related sections within this thesis. 96
13. An updated overview of The Planning Game. 100
14. Updated story estimation. 101
15. The planning game staff completion. 112
16. Sample system metaphor card. 122
17. The simple design practice within the XP process. 126
18. An outline of story evaluation planning. 133
19. The interconnectivity of the XP process. 176
20. The management flow of the XP process. 176

CHAPTER 1 – INTRODUCTION

The practical problem of creating software that is on time and within budget in a corporate environment can be a daunting task for any software project manager. The software project manager works with their people, processes, tools and measurements to accomplish this goal. During the late 1980's and throughout the 1990's, highly structured software development processes were created to address this practical problem. These processes took software development processes to one end of the spectrum by instantiating engineering-like control and measurements. The creation of the Extreme Programming (XP) process (1999 Beck) was a response to these engineering-based processes as XP focused on the other end of the spectrum of the software development process. XP focuses not on control and measurements, but on requirements, rapid development, and user input. While structured methods specifically addressed schedule accuracy, defect detection and correction, cost control, and adherence to completion dates, XP focused on requirements extraction, rapid development, and customer satisfaction with a more service based approach.

1.1– Problem Statement

While the XP process does not ignore schedule, cost, and quality, it fails to provide a management framework that can be used within a software organization that develops software for clients in strict contractual agreements. Budgetary bounds, quality standards, and specific deliverables are necessary elements of corporate contractual agreements. However, XP fails to provide the software project manager with the framework necessary to address these elements. When the schedule slips, costs are over/under budget, requirements are not being defined, and/or the projects stakeholders are unhappy or unaware of the current status of an XP project the XP process shortcomings can become contractual failure leading to litigation. The XP process needs a management framework to guide the XP process and provide accountability.

The goal of this thesis is to integrate standard software project management practices, where possible, into the XP process. Thus creating a management framework for XP project management that gives the XP managers the management activities and tools to utilize the XP process within a wider range of commercial computing organizations, relationships, and development projects. The framework will assess its ability to improve the XP process and identify its shortcomings.

1.2 – Approach/Methods

The approach used to develop the outline of practices and the approach to applying them was:

- Research all aspects of Extreme Programming (XP).
- Understand and define the 12 core practices of XP.
- Identify recent updates and modifications to these practices.
- Analyze how XP practices can be implemented in an organization.
- Analyze how XP's practices can be tracked, managed, predicted and used from a software project management perspective.
- Identify XP's weak points and criticisms, and then find ways to solve these issues utilizing software project management practices.
- Outline a basic structure for integrating software project management practices into XP.
- Identify the specific needs for each of these practices in conjunction with XP.
- Provide an example of how these software project management practices work in the real world when applied to XP.

1.3 – Software Process Background

The software process is a set of activities that every person or team completes to create a software product. Each of the activities can be summed up in four basic steps: define the product/software, create the product/software, test the product/software, and maintain the product/software. Every software project goes through all four of these basic steps. Different software process models specify the level of detail in each step and when each of these steps takes place. Examining the waterfall model and the spiral model, two established and widely used models, will provide a basic understanding of software processes on which our understanding of the XP model will be built.

The waterfall model (Figure 1) is one of the oldest and most widely used models. Introduced in 1970 by Royce, the waterfall model breaks the software development process into five categories: requirements and definition, system and software design, implementation and unit testing, integration and system testing, and operation and maintenance (Sommerville 2001, 46). The idea is that each phase must be completed before moving on to the next phase. Each phase can be reiterated, but each phase results in static documentation and is costly to re-work. The model works best when requirements are clearly defined and do not change throughout the process. However, in reality each step is not always completed before the next step starts.

The spiral software development model (Figure 2) was proposed by Boehm and has been used to address the time-to-market problem in software development (Boehm 1988, 64). The spiral model is defined as "a family of software development processes characterized by repeatedly iterating a set of elemental development processes and managing risk so it is actively being reduced" (Boehm 2000, vii).

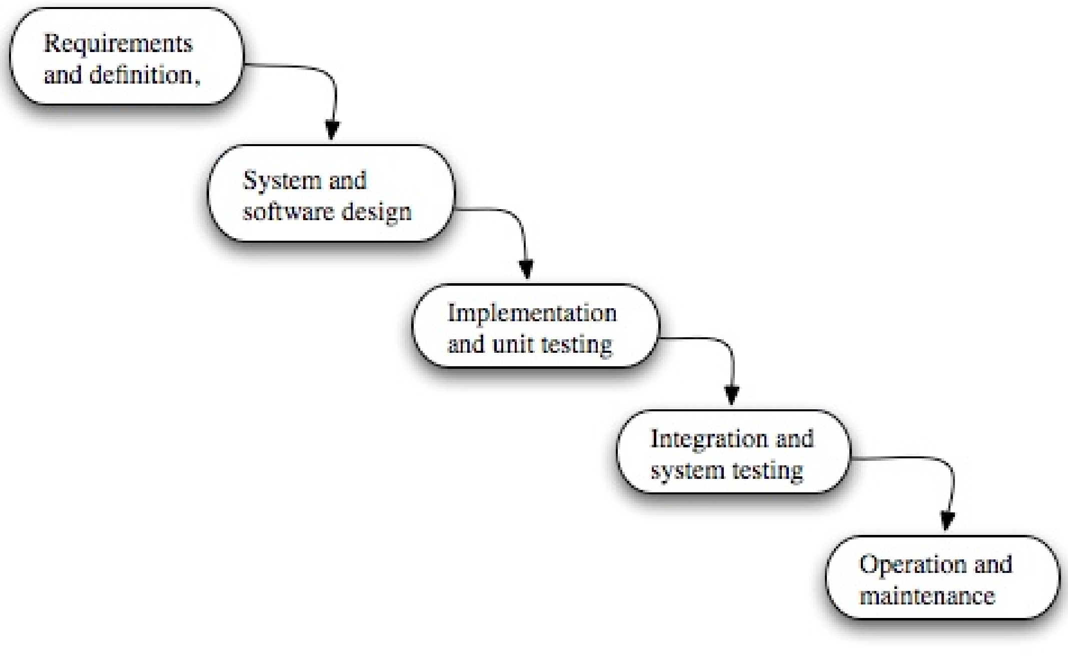

Requirements and definition.
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 1 – The waterfall model.

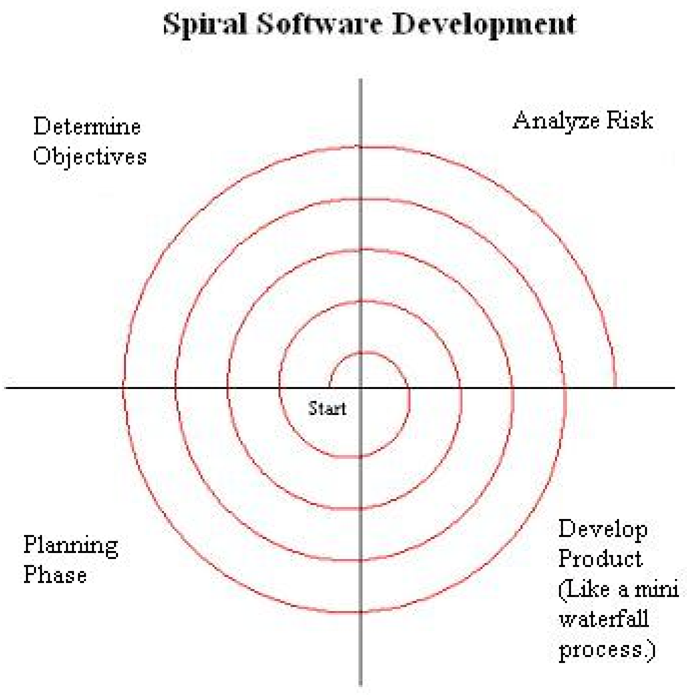

Spiral Software Development
Determine Objectives
Analyze Risk
Start
Planning Phase
Develop Product (Like a mini waterfall process.)

Figure 2 – The spiral model.

The process starts by determining the objective of the first iteration of the product then analyzing the risk associated with this iteration. A risk is anything that could hurt the project. The development phase is very similar to the waterfall model as the process includes the same five steps: design, code, testing, deployment, and service. The spiral model also includes a final planning phase where the project's current status is reviewed and planning for the next spiral begins; then the whole process starts over again. The spiral development process was the first software development process to directly address project risks and try to minimize them.

The waterfall model and the spiral model are two types of software design processes. Both have their strengths and weaknesses and their place in software engineering. When examining XP and how to manage a project, it is important to note the differences between these development processes. On a continuum from predictive to adaptive software design processes (Figure 3), XP is highly adaptive where the waterfall model is more predictive and the spiral model is more insulated from risk.

The purpose of this thesis is to define a simplified outline of best practices and a realistic approach for software project management using an XP software process. A framework for implementing XP will be created by using and expanding on current and alternative software development models. To this point, this thesis has laid down the foundation for this goal. The rest of the thesis will focus on the author's interpretation of XP and how to lead a software project using a modified view of the process.

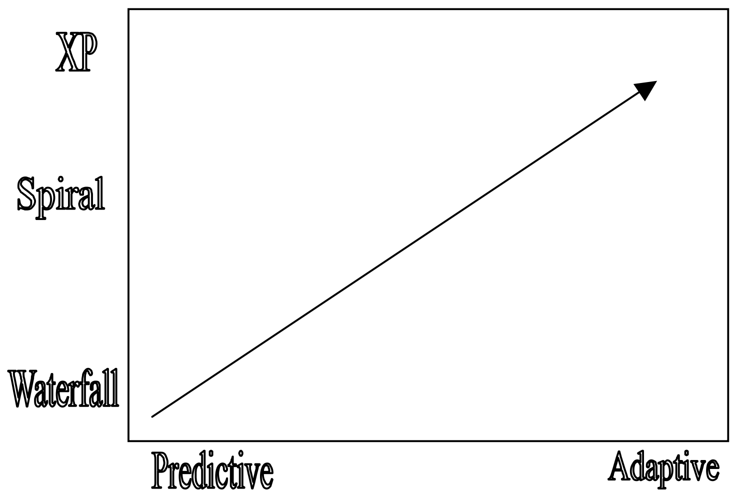


Figure 3 – A comparison of software design models.

CHAPTER 2 – OVERVIEW OF EXTREME PROGRAMMING

Kent Beck, the creator of XP, defines the process as “a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software” (Beck 2000, xvii). Ron Jefferies provides an updated definition of XP as:

> “a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.” (Jeffries 2001)

The author views XP as a software development process created out of necessity and reality. Even with high quality personnel, the reality is that the best software engineers following software engineering principals still fall short of a true software engineering discipline. To continue this discussion, it is necessary to define software engineering.

Software engineering is “the application of scientifically based engineering principals and practices in order to build high quality software on time and within budget” (Henry 2003, CS 346 Lecture). Extreme programming does not focus on these “scientifically based engineering principals,” instead it focuses on building high quality software on time and within budget. XP is a blend of proven software development principals and an understanding that building software is still a craft. A craft that best performed by quality individuals working in a relatively unstructured environment.

By focusing on simplicity, communication, feedback, and courage, XP’s real asset is people. The software engineers that work in an XP process work within the framework of 12 guidelines that define their process: the planning game, small releases, a system metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-hour work week, an on-site client, and strong coding standards.

2.1 – The Purpose of XP

Extreme programming was born out of a necessity to limit or shield software projects from risk. Beck states that, "software development fails to deliver, and fails to deliver value. This failure has huge economic and human impact. We need to find a way to develop software" (Beck 2000, 1). Beck identifies areas of risk that XP tries to limit or minimize including: schedule slips, project cancellation, defect rate, a misunderstood problem, business changes, the dancing bear problem, and staff turnover (Beck 2000, 1). All of these areas of risk can exert influence throughout the lifecycle of a project. Of the ten listed risks, most can be associated with change during the software process. Change at any level of software development increases the inherent risk associated with that stage in the development process. One of the problems associated with these defined risks is cost. It has been known for a long time that in software development the cost of change during the development cycle has an exponential relationship with the length of time the change lies incomplete during the software development schedule (Figure 4).

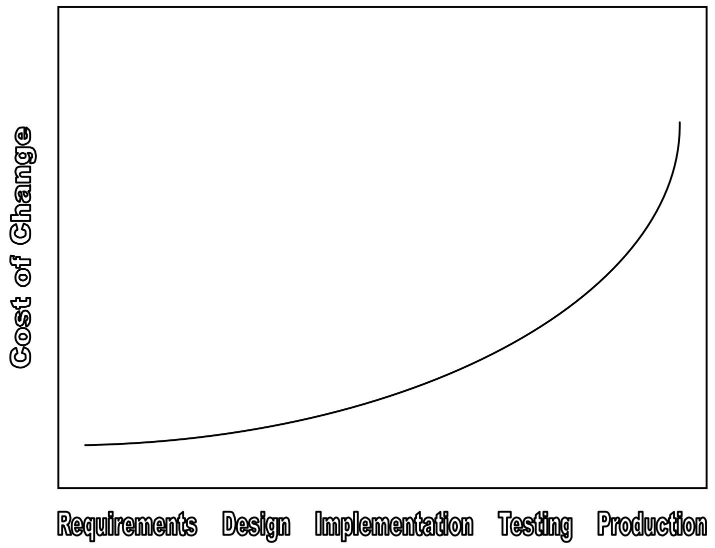


Figure 4 – Cost of process changes.

2.2 – Rights Within XP

One of the cornerstones of XP is the definition of the rights of the project manager, the rights of the clients, and the rights of the programmer. The concept of rights for all the parties involved is good; however the terminology could be improved. This terminology implies a balance between management, clients, and programmers. XP removes some of the project manager's responsibilities and delegates them to the clients and further asks the programming staff to be more than your typical programmers. For this to be effective, two things must happen. First, all the players on the team must understand their position and role. Second, everyone needs to be held accountable for their responsibilities. The process of defining these "rights" effectively creates a contract for all the team members and lays out a general path for the project.

The client is expected to choose which element will deliver the most business value, which functionalities get created, and in what order (Jeffries, Anderson, and Hendrickson 2001, 59). The client then creates acceptance tests to show that the system does what they expect it to do. This gives the clients a feeling of joint ownership. This process also recognizes that outside political, structural, and cultural problems with the clients' company could affect the overall outcome of the project. The managers and programmers could describe their association with the client best by quoting from the movie Jerry Maguire: "help me help you."

XP congregates the programmers into a team. The team is responsible for analyzing, designing, testing, and deploying deliverables. The team must accurately estimate the difficultly of all of the stories. In XP, stories are written by the clients to explain a desired functionality. The programmers then estimate how long each of the stories would take to implement and return these estimates to the client. The XP philosophy is to emphasize software development. The software project manager uses this emphasis to focus the team on constantly adding business value for the client and the company.

In an XP project, the software project managers' job is to keep the software development process moving forward. They must keep the lines of communication flowing to and from the clients. In theory, an XP process has very few wasted activities and the process always results in more value to the project. Keeping the process flowing smoothly is the most efficient path to completion. Jeffries believes that the manager's role is not to participate in the process, but make the process smoother (Jeffries, Anderson, and Hendrickson 2001, 4). The framework provided in this thesis will vary from this belief.

Once the roles of the project manager, the client, and the programming staff are defined XP reinforces the relationship by defining a bill of rights. The rights of the manager/client and the programmers are common sense. These rights are listed below as a hand-shake contract that all parties agree on when they enter an XP project.

Manager and Client Rights (Jeffries, Anderson, and Hendrickson 2001, 7)

- You have the right to an overall plan, to know what can be accomplished, when and at what cost.
- You have the right to get the most value out of every programming week.
- You have the right to see progress in a running system, proven to work by passing repeatable tests that you specify.
- You have the right to change your mind, to substitute functionality, and to change priorities without paying exorbitant costs.
- You have the right to be informed of schedule changes, in time to choose how to reduce scope to restore the original date.
- You have the right to cancel at any time and be left with a useful working system reflecting investment to date.

Programmer Rights (Jeffries, Anderson, and Hendrickson 2001, 7)

- You have the right to know what is needed, with clear declarations of priority.
- You have the right to produce quality work at all times.
- You have the right to ask for and receive help from peers, superiors, and clients.

- You have the right to make and update your own estimates.
- You have the right to accept your responsibilities instead of having them assigned to you.

2.3 – XP Core Values

A value can be defined as a principle, standard, or quality considered worthwhile or desirable (Webster's 1996, 1473). XP relies upon four core values: communication, simplicity, feedback, and courage. These values represent the standard that all the members of the XP team strive to maintain. These values are not meant to be a process that the XP team follows, but rather to provide an ethical compass. The 12 practices of XP continually use these core values with the principles that guide the development process.

Communication is a key component in every step of the XP process. For the development process to be as agile as possible XP relies heavily on verbal communication. This is very different from document-driven processes such as the waterfall model. Much of the day-to-day operations of an XP process rely on defining, overhearing, or asking questions. It is critical to have all members of an XP team talking to each other, their manager, and the client.

Simplicity is one of the most valuable ideas in the XP process. For every feature or activity, the XP process asks: "What is the simplest thing that could possibly work?" (Beck 2000, 30) This is a radical change from most object oriented design processes. Most developers will try to predict the future and code in a way that promotes a very adaptable solution. The accepted wisdom is that software always changes and the programmer should try to program in a way that can handle these changes. The problem is, no one can predict the future and it is pointless to try. The XP process believes that the programmers should code in a way that solves the current problem. If the solution is in its simplest form, then changing it in the future will be less painful, and the team didn't waist time trying to solve an unknown problem (Jeffries, Anderson, and Hendrickson 2001, 15).

Feedback is used in the XP process to make small adjustments to the course of the project. Schedule slips and missed deadlines happen an hour at a time. XP does not wait for the schedule to get a week or more behind. It takes a proactive approach by continually getting feedback from the team. The clients help steer the project by always knowing its status, and are available for even the smallest of questions. In turn, the programmers can always provide an up-to-the-minute status report for the clients. In broad terms, feedback is constantly provided to all the members of the team.

The last value of XP is courage. Courage is one of the hardest elements in programming. It takes courage to admit when you're wrong, especially when the consequences of your mistake are visible in the code. Refactoring can be scary and it can impose more work on the team. Shortsighted team members may be tempted to take the easy road and push off problems that someone else may need to solve. XP is a process where the programmers work in the now, and often need to have the courage to make changes to the existing code base as problems appear.

2.4 – XP Core Practices

The four core values of XP – simplicity, communication, feedback, and courage – can be embodied and defined within a set of 12 principals that guide the software development process. This section will define those 12 principals and subsequent sections will add insight on how a software project manager can take these principals from paper to practice.

Since 2000, when Beck first published his book *Extreme Programming Explained*, extreme programming has undergone few changes. Beck, Jeffries and Wake all feel that the 12 principals of XP are interconnected and if you're not using all of the 12 principals, the process is not extreme programming. This thesis will take a step back from this very strict adherence to the 12 principals. Thus, this thesis will portray the 12 principals with as much flexibility as possible and allow the project manager to adjust for specific needs. The 12 principals of extreme programming are: the planning game, small releases, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, on-site client, coding standards, 40 work week, and system metaphor.

2.5 – The Planning Game

The planning game is a series of activities between the programmers and the clients that define what features will be implemented in the next iteration of the software. In this process the client defines a list of stories that need to be implemented in the software. The client also prioritizes these stories based on time estimates from the programmers. Then the clients and the programmers increase/decrease the number of stories per iteration/release until the number falls within the range that the programmers say can be accomplished. The planning game is very similar for iteration planning and release planning. Both involve story cards, story estimation, the clients, the programmers, and a lot of communication. This section will define and explain user stories, how to estimate them, and then apply the planning game to release and iteration planning. Figure 5 is a flow chart showing the basic steps in the planning game.

2.5.1 – User Stories

User stories in the XP process are the system requirements. The difference is that stories are created and edited by the clients. The stories themselves are a short description of the behavior of the system that the user will see and use (Jeffries, Anderson, and Hendrickson 2001, 24). Focusing on the user's perspective is a major difference from formal requirements that focus on specifics of what the system needs to accomplish. User stories are similar to use cases but they focus on the goal that the desired functionality encompasses.

User stories are literally written on 5x8 unlined cards (Jeffries, Anderson, and Hendrickson 2001, 24). All of the stories are written by the clients. The stories themselves are comprised of a specific instance about how the system will be used. While the clients write the stories, the programmers listen and ask questions. One user story fits on one side of the 5x8 card, if it doesn't the clients simplify the story down to the core of the problem. Also, the stories should be no more than one week worth of programming work. If they encompass more than one week worth of work, the programmers ask the clients to split the story into two or more stories. The goal is to make each story be its own chunk of work that can be easily estimated. The XP process believes that time estimates over one week worth of work are less accurate.

The client will explain and expand each story directly to the team, who will then implement it. Thus, the story cards do not have to contain very specific requirements. Figure 6 shows two examples of user stories.

Every major function in the system should have at least one story. Jeffries states that, "you'll probably need at least one story per programmer per month. Two would be better. So if you have ten people for six months, that's between 60 and 120 stories" (Jeffries, Anderson, and Hendrickson 2001, 88).

The story cards themselves are promises for a conversation. The conversation that takes place is between the programmers and the clients. This is done to clarify and add depth to each of the stories. The stories can be used as placeholders for additional documentation that is attached to a story. Examples include, CRC cards (Class Responsibility Collaborator), acceptance tests, and application code.

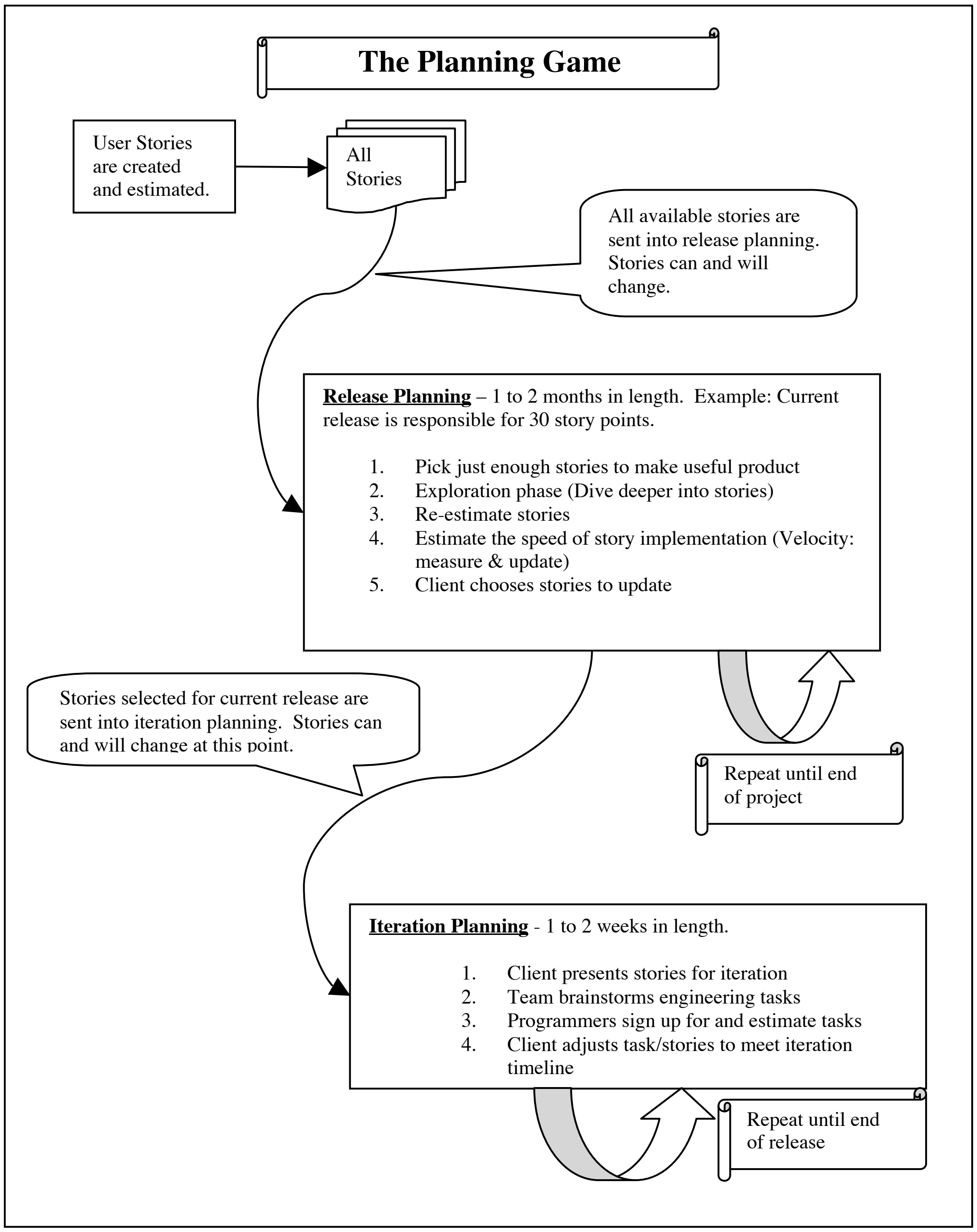

The Planning Game
User Stories are created and estimated.
All Stories
All available stories are sent into release planning. Stories can and will change.
Release Planning – 1 to 2 months in length. Example: Current release is responsible for 30 story points.
1. Pick just enough stories to make useful product
2. Exploration phase (Dive deeper into stories)
3. Re-estimate stories
4. Estimate the speed of story implementation (Velocity: measure & update)
5. Client chooses stories to update
Stories selected for current release are sent into iteration planning. Stories can and will change at this point.
Repeat until end of project
Iteration Planning - 1 to 2 weeks in length.
1. Client presents stories for iteration
2. Team brainstorms engineering tasks
3. Programmers sign up for and estimate tasks
4. Client adjusts task/stories to meet iteration timeline
Repeat until end of release

Figure 5 – The Planning Game.

XP project managers work on multiple projects a year. The application stores and provides easy access to specific information about any project. When the user chooses a project the system will show all the stories for that project.

The physical story card is important to capture by the system. The project manager wants the card stored in a digital manner, but be able to recreate the cards if needed.

Figure 6 – Sample user stories.

2.5.2 – Story Estimation

Story estimation is the XP way of assigning effort and time to system requirements. This process is done during the release and iteration planning phases of the XP process. It involves the clients, the programmers and three activities: spike solutions, story splits, and story points.

During iteration or release planning sessions the team of programmers and the clients gather. The clients start by writing, updating, or changing the user stories. Then the programmers review the stories and assign a point value to each story. Figure 7 shows a flow chart of the process.

A split in a story transpires when the programmers cannot estimate the whole story, or if the clients realize that one part of the story is more important than the rest, and they can split it into two or more stories (Beck 2000, 90). If a story appears to be larger than 1-2 weeks in size the clients are asked to split the story in two or more parts. On the other end of the spectrum, if the story isn't clear but the programmers can't estimate its length, or its time for the programming staff to spike a solution.

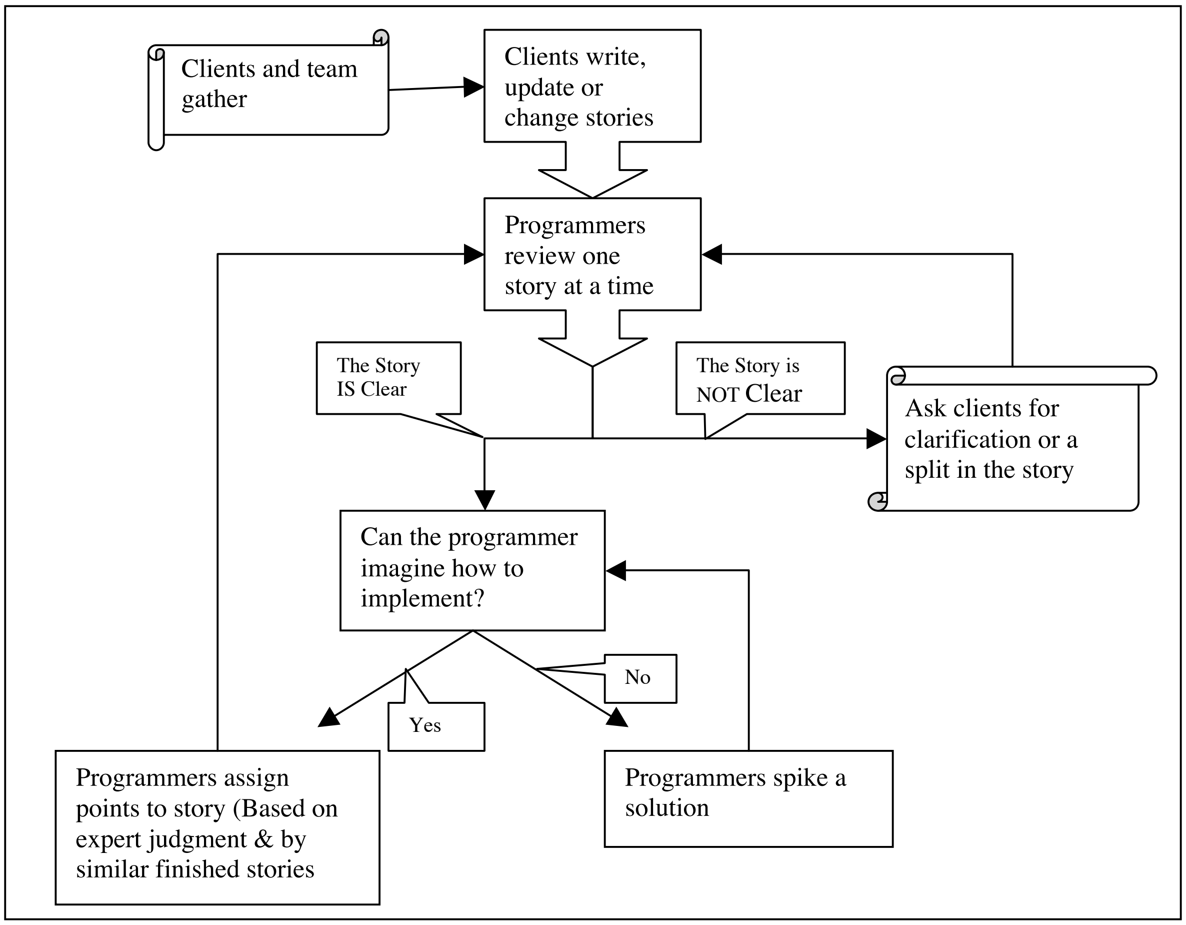

Clients and team gather
Clients write, update or change stories
Programmers review one story at a time
The Story IS Clear
The Story is NOT Clear
Ask clients for clarification or a split in the story
Can the programmer imagine how to implement?
Yes
No
Programmers assign points to story (Based on expert judgment & by similar finished stories
Programmers spike a solution

Figure 7 – Story estimation.

A spike is a small coding project that allows the programming team to imagine how a story is implemented. They are not meant to be the solution, but provide just enough information so the programming staff can estimate the time and effort for a story.

Last, XP measures time and effort by points assigned to specific stories. XP measures time and effort in perfect engineering weeks, one point equals one perfect engineering week. Perfect engineering weeks are defined for the programming staff as one week without any interruptions and the programmers code perfectly all day every day. This process is augmented by referencing past similar stories to predict points for the current story.

2.5.3 – Release Planning

Releases in XP are made up of iterations. Each of the releases is comprised of completed iterations, where implemented stories are taken as tasks by the programmers and completed. In XP the framework for planning a release and planning each iteration involves the same steps: exploration, commitment, and steering (Jeffries, Anderson, and Hendrickson 2001, 56). Each step adds detail and the ability for the clients and the programmers to adjust the plan and estimates.

Before release planning can begin the story estimation process has to be finished. This does not mean that all the stories for the finished product are defined and estimated. Stories will be added, deleted, and changed throughout the XP process. But at this point in time, all the stories that encompass the software to that point are completed and estimated. The steps for a release planning meeting are (Jeffries, Anderson, and Hendrickson 2001, 59):

1. Pick or write just enough stories to define a successful product. (1 – 2 months worth of stories);
2. Re-estimate stories (See section – 2.5.2);
3. Clients choose which stories to implement based on business value and difficulty.

These three steps will be done for every release planning session. Although the steps are similar, they do not encompass the same steps for the iteration planning process. The three phases below, exploration phase, commitment phase, and the steering phase are in a general format and apply to release and iteration planning.

2.5.3.1 – Exploration Phase—In the exploration phase the clients write or update the user stories. The programming staff asks questions and tries to understand each story in its entirety. If the stories involve more than one to two weeks worth or programming time, the client are asked to split the story. If the stories are clear, but the programming staff does not have enough information to assign an estimate to the story the team does a quick spike solution to gather more information.

The client gets to decide when they what the release date. They also get to decide what features or stories they want to include in the release. The programmers tell the client how many points they can accomplish in the given time period. This number is the projects velocity (Jeffries, Anderson, and Hendrickson 2001, 58).

XP uses the term velocity to represent how many story estimation points the team or individual can finish in a release or an iteration. The team tracks how many points they finished last iteration and last release to improve their time estimates for the clients. For example, the XP team may have finished 30 story points last release and for the next release the clients can pick as many stories in any order to implement as long as the total story points are less than or equal to 30.

In this phase each programmer will advise the team of how many points they can complete it the given time period. For example, the clients want the first release to be two months in length. Each programmer will write down the total number of story points they can complete in that timeline. Some programmers will be able to do more than others. The programmers then add up all of their estimated points and ask the clients to decide what stories get implemented in this release.

<u>2.5.3.2 – Commitment Phase</u>—Once the clients have an accurate estimation of the amount of story points that can be accomplished for the phase, release or iteration, they commit to the stories that will add the most value to the system and add up to the correct number of story points. At this point the programmers commit to accomplishing those stories or tasks, for this phase of iteration planning.

<u>2.5.3.3 – Steering Phase</u>—The purpose of the steering phase is to update the plan based on what is learned by the programmers and clients. Because all the stories and tasks are well defined, when an objective is not being accomplished the client and programmers already have the tools they need to adjust scope and re-evaluate the team's velocity. Release planning gives the clients more control on directing what gets

implemented at a smaller scale (one to two months) and in an XP process this is how the clients steer the product from release to release.

2.5.4 – Iteration Planning

When the iteration planning process starts the clients have decided what stories or features they want in the current release. This is how the clients direct the team, now the programming team needs to make adjustments to meet the client's goals; the first part of this process is iteration planning. Iteration planning has four steps (Jeffries, Anderson, and Hendrickson 2001, 63):

1. Client presents stories for iteration.
2. Team brainstorms engineering tasks.
3. Programmers sign up for and estimate tasks.
4. Client adjusts task/stories to meet iteration timeline.

In the release planning phase the programming team defined how many story points could be accomplished in the desired timeline. They worked with the clients to create, update, and estimate each user story. Then the clients had to decide what stories would add value to the product now and pick a number of stories that were within the maximum number of story points available.

Iteration planning takes the same process for the clients and applies it to the programming staff. First take a large problem (the release) and breaks it down into parts that can be estimated. Then split those parts among the team members and make adjustments to meet the release goals.

To start an iteration planning session the client reviews all the stories for the team to accomplish for this release. For example, the release may encompass 30 story points and the currently the team velocity is 10 story points per iteration. The client will add any details that the programming team needs to break the stories down into programming tasks. These tasks will also get assigned a story point value during a brainstorming session. "The team brainstorms the engineering tasks to build a common picture of the system design, down to the detail necessary to implement the stories" (Jeffries, Anderson, and Hendrickson 2001, 63). After all the stories are understood and broken down into tasks that have story points assigned, the commitment phase can begin.

In the commitment phase of iteration planning, the clients assign the correct number of stories to be completed for the iteration. Then the programmers sign up and take responsibility for the engineering tasks. Programmers can mix and match tasks between stories but one of the programmers has to be responsible for the whole story. Generally the team members sign up for all the tasks in a story. When the programmers sign up for a story they are allowed to ask for help from any of the other programmers. They have to say yes – it is a rule (Jeffries, Anderson, and Hendrickson 2001, 66).

The planning game works in three parts: story estimation, release planning, and iteration planning. All of these parts work together to discover requirements and result in rapid development.

2.6 – Small Releases

To maintain a high level of communication and feedback, an XP team requires frequent small releases of working software. Each of these releases can and will be evaluated by the client. This process allows the client to decide, based on the current system, which feature provides the most business value. The releases consist of a complete system, not just the final system. It is easier to plan and schedule one to two months at a time rather than six months. The practice of creating small releases is a methodology that makes sure the clients and/or management do not find themselves in the dark.

2.7 – System Metaphor

A metaphor is defined as "the application of a word or phrase to an object or concept it does not literally denote" (Webster's 1996, 851). XP teams use a system metaphor to add more to the requirements and align the team towards the same goal. Metaphors can provide common vision and feeling to the system. "This system keeps data more secure than a baby in a mothers womb." This statement could be an example of a system metaphor describing a biometric system for accessing a secure database. The system metaphor gets at the core of the project and aligns the desired features in a simple sentence.

Beck and Jeffries stress the importance of having a system metaphor but provide no guidance to how or when to create it. It is implied that the team creates it at the beginning of a project.

2.8 – Simple Design

Put simply, design and implement only what is needed today. Never "implement for today, design for tomorrow"(Beck 2000, 57). This idea goes against the main stream of software engineering, where programmers try to predict and design for change. The XP team always implements the simplest thing that could possibly work. If the system is simple, changing it to accommodate more requirements has less risk. Good design in the XP process (Jeffries, Anderson, and Hendrickson 2001, 76):

1. Runs all the tests (system test).
2. Has no duplicated logic.
3. Clearly states the programmer's intentions.
4. Has only the number of classes and methods necessary.

XP does not ignore system design it just wants the design to be simple. During the planning game an iteration planning team can use more formal design processes for specific tasks, one of the suggestion are to use CRC (Class Responsibility Cards) cards. The XP process does not endorse any specific design process and leaves much of the design of the system to the refactoring process.

2.9 – Testing

Ron Jefferies summarized XP's testing strategy as follows, "We will write tests before we code, minute by minute. We will preserve these tests forever, and run them all together frequently. We will also derive tests from the customer's perspective" (Beck 2000, 115).

XP requires the programming staff to create unit tests for everything that could possibly break. Unit tests are created first, for each unit of code, before the actual code is written. This test-first style of programming finds potential errors early, produces more stable code, and proves that the code does what it was designed to do. Unit testing gives confidence to the programmers that the software is behaving as desired and the clients can see that the inter workings of the software are functioning as planned.

The clients are responsible for writing acceptance tests. These tests validate the system by testing if the program performs its intended purpose. They help define the system and provide a functionality checklist for the programmers. Both the clients and the programming staff can see the current state of the product. Both unit tests and acceptance tests are part of the day-to-day activities of an XP project.

2.9.1 – Unit Tests

Unit tests in the XP process are test that must be repeatable and integrated as part of the system. Testing software commonly used with XP are JUnit and NUnit. Many of the IDEs available today, such as Visual Studio 2005, Eclipse, and Netbeans, include unit testing software as part of the development environment. These software frameworks allow the programmer to write testing code that can be run at anytime and is part of the system forever.

The XP process lays out rules for unit tests. Below is a list of rules for creating unit tests as defined by Beck (Beck 2000, 116):

- Each test doesn't interact with other tests in the system, they stand on their own.
- The tests are automatic.
- Test everything that could break.
- Tests are written method-by-method.
- Unit tests are bets that the programmers are making that the code will succeed when it should fail or that it fails when it should succeed.
- Programmer-written unit tests always run 100 percent.

Of the above list, the most important rule is the last one. No code gets released until all of the unit tests pass. If a test is failing and the programmers who wrote it cannot fix it, fixing the test becomes the high priority of the XP team. This is done to have a baseline of working functionality that the clients and the programmers can trust.

The XP process also lays out a framework for deciding when to create unit test. This framework is meant to guide XP teams, but not to be hard and fast rules. When to write tests (Beck 2000, 117):

- If the interface for a method is at all unclear, you write a test before you write the method.
- If the interface is clear, but you imagine that the implementation will be the least bit complicated, you write a test before you write the method.
- If you think of an unusual circumstance in which the code should work as written, you write a test to communicate the circumstance.
- If you find a problem later, you write a test that isolates the problem.
- If you are about to refactor some code, you aren't sure how it is supposed to behave, and there isn't already a test for the aspect of the behavior in question, you write the test.

The process of writing unit test is a simple one, during a coding session the team of programmers starts by (Jeffries, Anderson, and Hendrickson 2001, 98):

1. Creating a test class for the functionality to be tested.
2. Create a setup method for the code to be tested and any variables or parameters that are needed for the setup of the testing code.

3. Write the test in the test class. Write the necessary stub methods in the code base to be able to run a test.
4. Run the test, it should fail at this point. The desired method or class has just stub methods and no real code behind them, just enough to create the test.
5. Implement the class or method that the test is written for.
6. Run the test, if the test passes then release the code. If not, fix the bug.

The goal of XP units is to give the programmer the confidence and tools to maintain, update and create a large body of code. This confidence come from knowing that if the programmer ventures into a section of the code they are unfamiliar with or change the code, tests are in place to verify that no damage has be done and the additions are working.

2.9.2 – Acceptance Tests

Acceptance tests are XP's way of proving the system works to the clients and redefining what the programming staff needs to accomplish. Acceptance tests are created throughout the XP process. The sooner the XP team can catch a mistake, the sooner they can make the program work. The clients, with the help of the programming staff, create repeatable tests for every story they have the XP team implement.

- The customers write tests story-by-story.
- An XP team of any size carries at least one dedicated tester to help the clients create acceptance tests.
- Tests should be automated.

Examples of acceptance tests include: a batch file program, JUnit, NUnit tests, spreadsheets, or file based tests. What is important is that every story gets tested and they are automated. The XP process defines why acceptance tests are important and that they are a critical part of the XP process. Where Beck and Jeffries are unclear is how to implement acceptance tests. Because the process is a framework for creating software and new languages, tools, and products change constantly XP leave the implementation up to the XP team itself.

2.10 – Refactoring

Fowler defines refactoring as “a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior” Fowler 2007). Using courage and past unit tests makes restructuring manageable. XP teams work toward the “once and only once” principle to coding, and try to reuse as much as possible. Refactoring is part of the XP culture because programmers do not try to predict and code for the future. They make the code simple and have a willingness to change the design when it becomes necessary.

Refactoring is XP’s way of simplifying the software design process and specifying a way to change the code without rewriting the system. First on the initial design, XP teams only implement the simplest design that will work. Beck defines the simplest design by four constraints, in priority order (Beck 2000, 109).

1. The system must communicate everything you want to communicate.
2. The system must contain no duplicate code.
3. The system should have the fewest possible classes.
4. The system should have the fewest possible methods.

Given these four constraints of simplicity, the design process for an XP team is simple. First the programmers start with a test to define what is going to be accomplished. At his point a little traditional design is implemented; what are the classes and methods involved. Next make sure to implement just enough to run the test. The test defines what is going to be accomplished and the programmers need only to work towards that goal. Then, repeat steps one and two. Lastly, if the programmers see an opportunity to improve the code, they do it, but within the four simple constraints. These steps define XP’s design and refactoring process.

2.11 – Pair Programming

Pair programming is the most controversial practice of XP. This practice differs from every other programming methodology. As the name implies, all of the programming is done in pairs. One of the programmers sits at the keyboard and types the code, while the other programmer thinks critically about what they are coding. Each programmer switches between programmer and verbal architect. The amount of time each person spends programming versus thinking critically about the code is not important per say; both programmers must be mentally active in the process. Pair programming produces fewer defects, more efficient code, and increases overall knowledge of the code (Jeffries, Anderson, and Hendrickson 2001, 78).

In the XP process, pair programming is designed to create the best quality code as efficiently as possible. Pair programming is not meant to be a tutoring session for an inexperienced programmer. Neither is it one person coding while the other person watches. The goal of pair programming sessions is to form a mind-meld between two programmers to create high quality code quickly. This is done by having rules for the pairs. First, programmers are not stuck with each other. A team may pair in the morning and split to pair with other people in the afternoon. Second, roles are divided in two categories: the driver and the partner.

The driver's role is to sit at the keyboard and type in the code (Beck 2000, 89). They are the architect and designer. When they are driving they have the final decision on implementation decisions including classes, methods, and the flow of the program. They try to create meaningful code that can be understood now, months from now, or years from now. They rely on their partner to support them in this endeavor; after all, they will soon be the partner. The driver is responsible for keeping their partner completely engaged in the session. Communication is a key aspect for the driver. They have to tell their partner what the methods and classes are going to accomplish. This is done so the partner can alert the driver of potential pitfalls, code improvements, alternative options, and act as a sounding board.

During the pair programming sessions the partner's job is to support the driver. They keep track of objects, methods and system design so the driver can focus on the task at hand. It important that the partner not just sit back and watch the driver code. At any time they can say, "let me drive" and then they become the driver. To be effective the partner has to be mentally ready at any time to take over where the driver left off.

Pair programming takes work by both developers. It can be a balancing act between communication and execution. XP acknowledges that pair programming produces more code that has fewer defects. However, used within the XP process, it also results in the team having a better understanding of the code in a more enjoyable coding environment (Beck 2000, 90). One additional benefit of the pair programming process is that it enables and supports the other practices of XP; creating unit tests, communicating with the client, refactoring, and following a coding standard can sometime are forgotten under stress. Software projects are often over budget and past due, having another developer sitting with the driver lessens the chances they both will cut corners and ignore the process (Beck 2000, 102).

2.12 – Collective Ownership

No one person in an XP team "owns" any given part of the code. Everyone in the team is responsible for all parts of the program. In most non-XP projects the programming staff is grouped by specialty. Traditional development teams have the members grouped by their areas of expertise, e.g. database expert, web specialist, report writer, etc. This never happens in an XP team. The programming pairs are required to work on areas outside of their expertise. This increases the overall knowledge of the team, while decreasing the potential bottleneck of a specialist. In short, anyone can work on any part of the program.

2.13 – Continuous Integration

An XP team should always be adding business value to the product. During production, the software is maintained in a deliverable state at all times. A deliverable state means that the software is releasable and could be installed; it does not mean that the program is completed. The idea is to maximize the efforts of the team and allow management and clients to provide almost real-time feedback.

Continuous updating of the current software to a releasable state is what an XP team calls continuous integration. A test machine is maintained for integrating new code. After a few hours of programming, the programmers take their changes to the test machine, add or merge their improvements, and run all of the unit tests. When 100 percent of the tests run successfully, they add their code to the release version, build the system and begin work on the next story.

System builds should happen many times a day (Jeffries, Anderson, and Hendrickson 2001, 78). Because the team is releasing and integrating their changes to the system many times a day it is important to mindful of build times. If the build gets too large the team can spend too much time waiting for the system to compile instead of working on new functionality. Jeffries recommends (Jeffries, Anderson, and Hendrickson 2001, 78):

1. Make build scripts to keep as many files as possible in object form except when they are edited.
2. Use dynamic loading.
3. Build intermediate partially linked objects files.
4. Build lots of DLLs when possible.

2.13.1 – Releasing Changes

Continuous integration uses code ownership and comprehensive unit testing to release code rapidly and reliably. The process of releasing changes is a three step process (Jeffries, Anderson, and Hendrickson 2001, 122):

1. Get the current release and code locally.
2. Once the team has finished their task, they begin the updating the current release.
3. Release to the current release machine. The process is done when all the unit tests run on the release machine

The XP process recommends using code management software tools for development. The benefits of these tools are well known and will not be discussed here. The process does not specially recommend any particular code management system; just have one that fits the need of the project. The XP process does have a couple of rule how to use the code management tools (Jeffries, Anderson, and Hendrickson 2001, 124). First, set as few restrictions as possible. No individual users or passwords. Second, when the project starts there should not be segmentation into code groups or topics. Start with a flat structure and let the development process create the structure of the software.

2.13.2 – Possible Problems and Fixes

Releasing code in an XP process can be problematic. Two major problems that can occur are slow merges and lost changes. This section will examine each and offer solutions.

2.13.3 – Slow Merges

Slow merges can occur when there are conflicts in edits. This can happen more often at the beginning of an XP project because the team does not pre-design the system or divide the classes, methods, or files into manageable parts. These divisions happen naturally when the user stories start to get implemented. Beck recommends that slow merges should not be fixed by allowing simultaneously editing of the code (Jeffries, Anderson, and Hendrickson 2001, 124). The XP process wants the code lead the system not merge conflicts. To combat merge conflicts, the XP team should integrate their changes as much as possible.

2.13.4 – Lost Changes

Lost changes can occur due to incorrectly merged code (Jeffries, Anderson, and Hendrickson 2001, 125). These happen normally because a conflicting edit was not recognized or the team incorrectly merged two sets of changes. Lost changes can be solved with three steps (Jeffries, Anderson, and Hendrickson 2001, 125):

1. Release changes frequently.
2. If loss happens, the system test will help find it.
3. Backup the source code.

2.14 – 40-Hour Workweek

The XP process is fast, responsive, and intense. It demands a lot from its programmers, managers, and clients. All of the participants must work at their highest level. The XP process recognizes that creating software is difficult and that tired team members create more defects. To produce the best possible product, an XP team works, on average, 40 hours in a week. The 40-hour week is not a rule, sometimes it is more, and sometimes it is less. Thus, the important point is that the team must be fresh and creative. No team member works over 60 hours in a week, and the team stays very close to 40 hours every week. The XP team works at a sustainable pace and focuses on achieving its end goal.

2.15 – On-Site Client

All of the practices of XP are geared toward having questions answered quickly with fewer barriers to communication. Having an on-site client enables every member of the team to go directly to the client and have any question addressed immediately. Recall that the client "chooses what will deliver business value, chooses what to do first and what to defer, and defines the test to show that the system does what it needs to" (Jeffries, Anderson, and Hendrickson 2001, 2). An XP team makes changes and improvements to the software on an hourly basis. In order to continuously adjust the course of the project an XP team requires the client to be a part of the process. The client should be a real user of the system.

Every programmer wants software requirements to be 100 percent complete and 100 percent correct; this is never the case. Requirement documents focus on what the software needs to do, not on why it needs to have a specific functionality. A real user of the system can provide the programmers with the "why," examples of workflow, and greater details for the requirement. This greatly increases the effectiveness of programming staff. The XP team needs to recognize that the on-site customer will have another job to do and they need to make the clients space as accommodating for the client's primary job as possible. Having a real user of the system on-site is critical to the success or failure of an XP project. If an on-site customer is not possible XP lays out five steps to minimize its impact (Jeffries, Anderson, and Hendrickson 2001, 21).

1. Try to get someone to represent the customer locally, preferably someone in the company who is an expert in the system area.
2. Try to get the real customer on-site for planning meetings.
3. Send the programmers to visit the customer as much as possible.
4. Frequently release your code to the customer.
5. Expect misunderstandings and prepare for them.

2.16 – Coding Standards

All of the programmers on an XP team must agree to and follow a concise coding standard. This standard must be maintained at all times and every team member must agree with the standard. With every programmer writing, refactoring, testing, designing, and planning small parts of the whole system, an XP team does not have time to learn specific coding styles for each part of the system. At the beginning of an XP project the team will agree on a shared coding standard. The specifics of the standard are not important, what is important is that all the team members can read, edit, and update the code. The coding standard gives all the team members a common understanding of how the code will be implemented.

2.17 – XP Practices Summary

One of the first things that drives the practices of XP is the interconnectivity of all of the 12 practices (Figure 8). The connections between all of the practices give the XP process its strength and validity. The XP process does not allow you to pick and choose which of the 12 practices you agree with and still have an XP process; the process requires all 12.

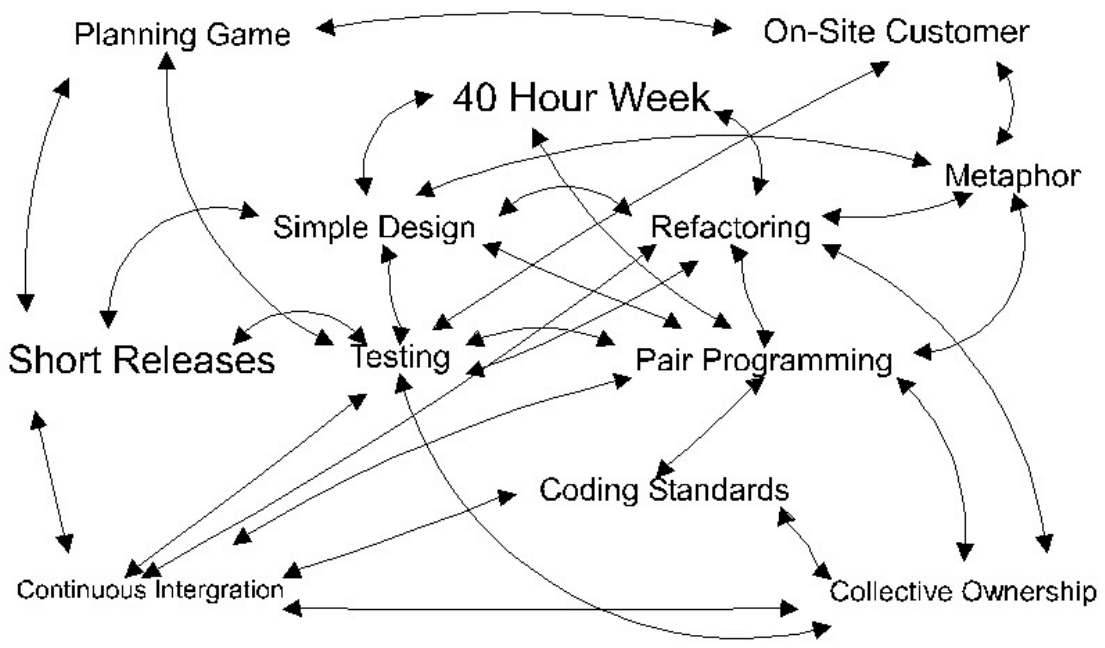


Figure 8 – Interconnectivity of the 12 core XP practices.

2.18 – Roles of the XP Team Members

Every member of an XP team has a role and set duties. Beck makes the analogy for XP roles to a sports team. Every sports team has defined positions for all the players and the coach's job is to get the best effort out of their team. The XP process defines six possible roles for each team member: programmer, customer, tester, tracker, coach, consultant, and manager (Beck 2000, 139-148). Each of these jobs needs to be filled by an XP team member. The next section will examine each of these positions and explain their duties. These roles are to be considered more guidelines than hard rules. If the XP team members do not fit the roles, change the roles not the people.

2.18.1 – Programmer

The programmer in an XP team is more involved in the overall software development process, then just their programming tasks. In the XP process they have three main areas of focus, programming, communication, and shared methods.

Much of the programmer's time is spent working with programs, making them bigger, simpler, and faster (Beck 2000, 41). Not only do they work at creating the desired software, but they create testing software to verify the code's function. They break the code into small pieces so each piece can be individually tested and verified. During this process the programmers only add code that adds the most value to the customer and develop nothing that isn't valuable to the client. XP programmers get to pick what functionality they are going to implement and how it will be done. This comes with a price; they have to accurately estimate how long each user story will take to implement and be responsible for the functionality. Lastly, they work to implement a simple design, test their code, and refactor often.

In addition, XP programmers spend much of their time communicating with other people: the clients, themselves, and management. An XP programmer must be a good oral communicator as much of the XP process is based on face-to-face communication. For example, during the planning game the programming staff has to estimate the amount of time and effort each user story will take. To accurately accomplish this task the XP programmers have to be able to listen to the clients and ask appropriate questions to both the clients and the other team members. In addition, in the pair programming process both programmers are talking about what they are trying to accomplish. Ultimately all the members of the XP team have a right to know the current status of the software in each of its pieces; thus every team member has to be able to communicate with all the other members.

The final requirement of an XP programmer is the ability to use shared methodologies. They must shed the idea of code ownership and have a feeling of shared ownership of the whole system. XP programmers are expected to have courage in themselves and the process (Beck 2000, 142).

2.18.2 – Customer

The customer plays as much of a role in the XP process as do the programmers. They know what the program is going to do and what needs to be programmed. Writing and good communication skills are required. The customers are required to write all of the user stories and communicate their meaning to the rest of the XP team if needed. In addition, the clients need to be able to prioritize the user stories based on cost proposed by the programmers. They need to be able to say “this is more important than that,” “this much of this story is enough,” “these stories together are just enough” with confidence (Beck 2000, 143).

The clients are also required to write repeatable acceptance tests in order to sign off on completed functionality. This requires the customer to know what is important to test and what to check for. These are skills that the customer will likely need to learn. They should also have domain knowledge of the software problem that their project is trying to solve. Preferably, they will use the system when the project is completed. To be effective in creating user stories and guiding the team, the client has to be able to look past how the problem was solved in the past and communicate what would add the most value to the system.

2.18.3 – Tester

Not much has been written or defined for the tester role since Beck initially proposed the XP process.

> "An XP tester is not a separate person, dedicated to breaking the system and humiliating the programmers. However, someone has to run all the test regularly (if you can't run your unit and functional tests together), broadcast test results, and to make sure that the testing tools run well." (Beck 2000, 144)

In the XP process the programmers create the unit tests and the customer creates the acceptance tests. This leaves the tester with the role of supporting those two processes and making sure the test results are known by the rest of the team.

2.18.4 – Tracker

The tracker for an XP team is responsible for being the conscience of the team (Beck 2000, 145). They gather, process, and report on all the system metrics for the project. Throughout the planning game the XP team makes a lot of estimates; the tracker records these estimates and periodically informs the team of their accuracy. They are also responsible for keeping an eye on the big picture. They should be able to tell the team if they are going to make the next release or if they need to make adjustments. All of the trackers measurements are kept as an historical record that can be used by subsequent project. While gathering their measurements the tracker should try not to interrupt the development process more than necessary.

2.18.5 – Coach

In the XP process the coach is responsible for the whole process (Beck 2000, 145). In an XP project all the team members are involved in the whole process, but the coach has to understand it more deeply completely and be responsible for the results. They focus on the process and the team. To allow for this additional responsibility the coach will tend to have fewer development tasks than other team members.

First, they monitor the process. They make sure the daily meetings are productive and not overly lengthy. They ensure that the team is creating the correct number of unit tests and that acceptance tests are getting written. They also monitor the people, making sure they are working together well, not leaving early or having general personnel issues.

Next, the coach enforces the process (Wake 2002, 140). The coach is responsible for taking action if any of the XP tasks are not being completed. If the rules of the XP process are not working the coach changes them to be more productive. They also mentor the team. Much of the XP process is based on small-interconnected activities and the coach helps the team members learn and expand their XP skills. They also interpret reports from the tracker about the current status of the project. The coach will review these reports and take the appropriate action. Overall, the coach manages the day-to-day operations of the XP team throughout the XP process.

2.18.6 – Consultant

From time to time in an XP project the team will not have the technical skills to tackle a problem. At this point the team has to hire a consultant to help the team overcome a hurdle.

More than likely the consultant will not be used to working in an XP process. To work with the consultant, the XP team must clearly define the problem to be solved (Beck 2000, 146). The team also needs to recognize that if they encountered this problem once, it may come up again. Thus, the team will work with the consultant to first solve the problem and second gain the necessary technical knowledge to solve it again. The XP team does this by working closely with the consultant and asking lots of questions.

2.18.7 – Manager

The XP manager brings together the customer and the programmers and facilitates the smooth operation of the process. They do not actually do the day-to-day activities of the XP process; they act to make the process smoother (Jeffries, Anderson, and Hendrickson 2001, 4).

The role of the XP manager is not to set priorities, assign tasks, estimate stories or dictate schedules. Those activities are divided up between the programmers and the customer. The major focus of an XP manager is to get things out of the way of the people who are doing the work (Jeffries, Anderson, and Hendrickson 2001, 5). This is done by removing everything out of the XP team's path that does not add to the objective of delivering quality software on-time. The manager's responsibilities can be listed as five tasks (Wake 2002, 132):

- Face outside parties
- Form the team
- Obtain resources
- Manage the team
- Manage problems

The XP process uses the manager as a shield from distractions. While the manager deals with all of the outside parties, the development team can continue without distractions. Outside parties can include the clients (the "funders" of the project), any internal personnel the team interacts with (e.g. System administrators, personnel department, etc.), and political problems that impact the team.

The manager also gets to hire, motivate, and train employees; handle annual review and salary adjustments; and correct undesired behavior with the team and/or possibly remove a member of the team. In short, the XP manager causes actions, coordinates the team, collects reports, gives rewards and removes obstacles (Jeffries, Anderson, and Hendrickson 2001, 6).

2.19 – XP Management Strategy

Where the previous section explored the roles of the team members, this section will expand on one of those roles; the XP project manager. The traditional software project manager's responsibilities can mainly be divided into two XP roles: the coach and the tracker (Beck 2000, 73). This section will examine just the XP manager's responsibilities and tasks. In addition, this section will add more depth to the coach and tracker positions.

Beck has defined some principles to guide XP project management (Beck 2000, 71):

- Accepted responsibility – It is the manager's job to draw attention to the tasks that need to be done, but not to delegate them.
- Quality work – The manager needs to trust the programming staff to produce a quality product. To do this, the manager removes any obstacles that inhibit this goal.
- Incremental change – The manager provides guidance all along.
- Local adaptation – The manager adjusts the practices of XP to fit the local culture or corporate environment.
- Travel light – The manager doesn't impose a lot of overhead.
- Honest measurement – Only use or gather measurements that report realistic levels of accuracy.

The XP style of management is more like decentralized decision making than centralized control. "The manager's job is to run the Planning Game, to collect metrics, to make sure the metrics are seen by those whose work is being measured, and occasionally to intervene in situations that can't be resolved in a distributed way" (Beck 2000, 72).

2.19.1 – Metrics

XP managers use basic metrics to guide and direct the team. Jefferies states that all the metrics used in an XP project should be no more complex than the simplest one that works (Jeffries, Anderson, and Hendrickson 2001, 137). Beck does not offer examples of specific metrics, but offers some guild lines (Beck 2000, 72):

- Metrics should be done with a big visible chart.
- Metrics loose their meaning over time, especially if they approach 100 percent. Search for an appropriate measurement.
- Metrics are a way of communicating the need for change.
- An XP manager's best tool is their feelings with respect to the project.

Jefferies offers four areas that XP metrics should focus on; resources, scope, quality and time (Jeffries, Anderson, and Hendrickson 2001, 136). XP metrics are used to understand the status of the XP team with respect of those four areas.

For the project resources, the XP manager wants to know: What are the project's assets and how were they used throughout the project (Jeffries, Anderson, and Hendrickson 2001, 136)? How many developers were involved in the project compared to how many were originally intended? How many clients were assigned or involved in the project? How many tests did the project need? What were the project's costs in terms of hardware, software, and management overhead?

Utilizing just one metric can do following the XP ideal of simplicity, tracking scope of the project. For each iteration of the project, update a bar chart of the number of story cards and how many are done. This chart can then help the manager determine:

- Are stories being split for better planning?
- Are stories being added? If so, is this because the team velocity is faster an expected, or does the project have a feature creep problem?

The next examples of metrics will focus on tracking and reporting quality. Tracking the number of acceptance tests (passed or failed) each month is a good measure of the scope of your testing (Figure 9) (Jeffries, Anderson, and Hendrickson 2001, 138).

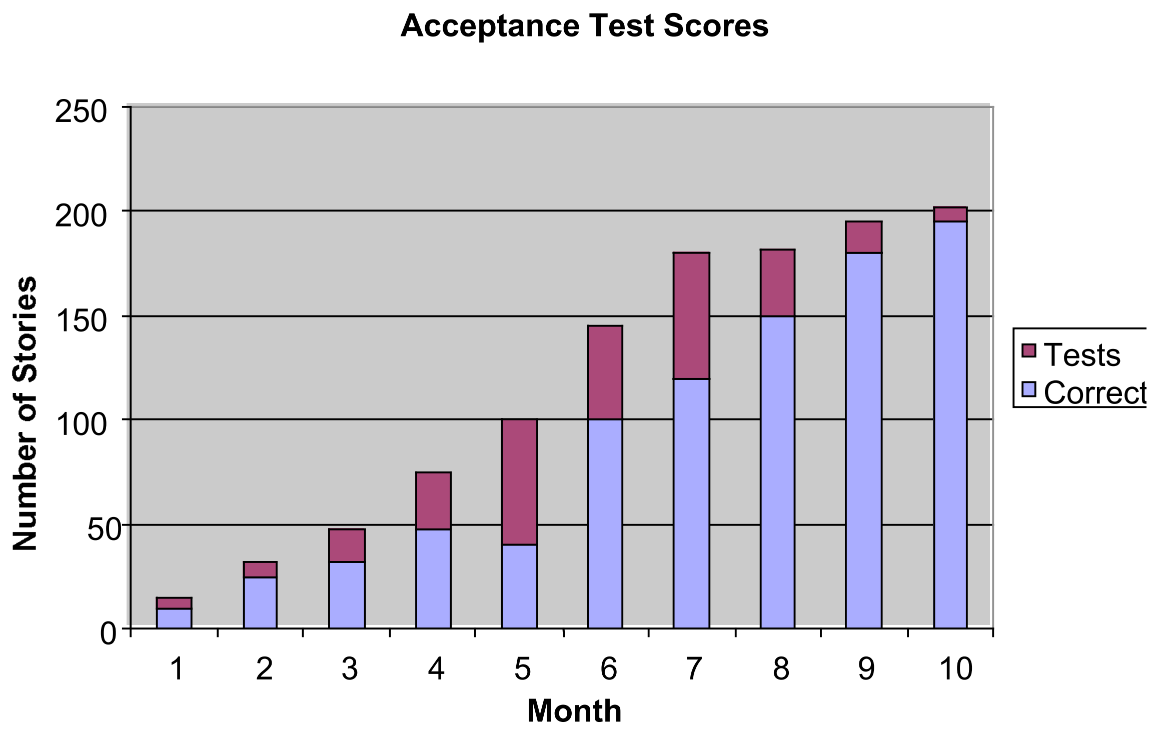


Figure 9 – Tracking acceptance tests.

The acceptance test scores will give the manager an idea of: How much has been done? Is the customer accepting the work? Is the number of stories leveling off or are they increasing? Are the acceptance tests catching up to the number of stories (tracking team velocity)? This report can be tracked daily, weekly and monthly. Sorting the test by program functions can also be helpful. This could shed some light on production problems in a specific part of the project.

The above examples show the types of simple measurement that the XP process relies on. Further examples include (Jeffries, Anderson, and Hendrickson 2001, 144):

- By date, the number of classes and methods.
- By date the number of test classes, methods and asserts.
- Total stories by date.
- Stories replaced, split or removed.
- Risk assessments.

- Meeting notes and observations.

2.19.2 – Coaching

The coach of the XP process was defined earlier (Section 2.18.5). This section will expand on how they are involved with XP management. The XP coach would in other teams be a lead architect or lead programmer. In the XP process the coach needs to have excellent technical skills, they should also have excellent communication skills. From a management perspective the coach has few development tasks (Beck 2000, 73). Their job is to get everyone making good decisions in the XP process. Beck defines the coaches' duties as follows (Beck 2000, 74):

- Be available during pair programming sessions.
- Watch the code; looking for long-term refactoring goals and encouraging small steps to reach these goals.
- Help the team with individual technical skills, e.g. testing, formatting or refactoring.
- Explain the XP process to the team and upper-level managers.

Two more duties of the coach are to acquire food and toys for the team (Beck 2000, 74).

2.19.3 – Intervention

Problems can occur that are beyond the teams ability to fix, this is when the XP manager has to intervene. The XP manager has to be a decision maker and be comfortable making tough choices. They should try to address problems early to minimize the effect on the team. Beck lists two areas where an XP manager has to intervene with the project: personnel-related issues and project-related problems (Beck 2000, 74).

Personnel problems can greatly affect the success of any XP project. The XP manager needs to address these problems early. They should look for any way the employee who is causing a problem can help the team. Once the solution is found, the manger needs to make the decision quickly and decisively, even if it is unpopular.

The manager also measures and reports the effectiveness of the process. If they see a need for change they act in an XP fashion. Beck says, “It isn’t the manager’s job to dictate what is to change and how, generally, but to point out the need for change. The team should come up with one or more experiments to run. Then the manager reports back on the measured changes caused by the experiment”(Beck 2000, 76). The last intervention an XP manager may need to recognize and perform is the early termination of the project.

2.20 – Facilities

Creating a work environment that is conducive to the XP process is the first step towards taking control of how the team works overall (Beck 2000, 80). XP facilities try to create an open workspace with small private spaces on the outside of a main room and the centerpiece being a common programming area in the middle of the room (Beck 2000, 77). The XP process relies on this type of work environment; it also contends that without a reasonable work environment the XP project will not be successful (Beck 2000, 78).

Beck suggested some simple facilities strategies (Beck 2000, 79):

- Error on the side of too much public space.
- Team members need to see each other, hear shouted questions, and “accidentally” hear conversations.
- Make the development area open and conducive for two developers to sit side by side for pair programming sessions.
- If cubicle wall are required, use half-height.
- Reserve a communal space off the side for the team.
- Separate the XP team from other teams in the organization.

2.21 – Steering

Steering is the XP way of making adjustments throughout the project. The XP process accepts that specifying and building software is difficult. The product is often changed due to inaccurate requirements, poor design, unrealistic schedules, and budget concerns. XP contends that most software project estimates are often wrong and priorities change before the project even begins (Jeffries, Anderson, and Hendrickson 2001, 148).

To combat these problems, XP teams steer themselves to success. The next two sections will discus iteration steering and release steering. Much of the steering process revolves around measuring the velocity of the team and planning to match that velocity. The XP process believes that selecting and adjusting the products functions to match the client's priority based on the team's current velocity, the XP team will deliver a better product that producing everything you originally planned (Jeffries, Anderson, and Hendrickson 2001, 149).

2.21.1 – Iteration Steering

During the iteration planning sessions (Section 2.5.4) the stories the clients created were broken down into programming tasks. The programming staff then signed up for and estimated these tasks based on how many days of effort they would take to accomplish. While executing each iteration the main goal is to complete all the stories (Jeffries, Anderson, and Hendrickson 2001, 152). The steering process focuses on completed stories, not partially implemented stories. Thus if the team has signed up for 10 stories and they can only complete nine in the allotted time, it better to have nine stories 100 percent completed than 10 stories 90 percent completed. The team and the clients can only count stories done when they are 100 percent completed.

The XP process relies on the accuracy of task estimation during the planning game. These estimates will not always be correct thus for the XP team to be successful, the team must learn from their mistakes and adjust as needed. To review, the XP team planned the iteration by (Jeffries, Anderson, and Hendrickson 2001, 152):

1. Breaking each story into tasks.
2. Each programmer signed up for and estimated these tasks.

Now the XP team needs to review and monitor those estimates. The goal is to improve the estimation accuracy at the story level in order to improve the chance of a successful project. The reviewing and monitoring of estimates in the XP process is called tracking. The tracking process identifies tasks that are exceeding their estimates and allows the team to deal with them (Jeffries, Anderson, and Hendrickson 2001, 153).

Every couple of days the teams tracks the process of the iteration at a task level. In the XP process this is done with face-to-face communication between the tracker and the individual programmers. It is not recommended that this process be done via email or otherwise automated (Jeffries, Anderson, and Hendrickson 2001,153). Team-building human contact is important in the XP process. The tracker needs four pieces of information (Jeffries, Anderson, and Hendrickson 2001, 153):

- All of the stories chosen for the iteration
- The tasks that need to be done
- Who signed up for each task
- The programmer's original estimates

Every few days the tracker asks the status for each task. It is important to find out if the task is ahead of schedule, on time, or behind schedule. Once the current status of the iteration is known, the tracker updates the team of its current progress.

When the tracker discoverers that the iteration is off schedule, the steps for the XP team are as follows (Jeffries, Anderson, and Hendrickson 2001, 153):

1. Bring all the information to the team level. If a story is having a problem the team has a problem.
2. Solve the story problem within the programming team. Try to make adjustments by reallocating team members who are ahead of schedule or others who could buckle down.

3. Give some special attention to the particular task that's off track. Call a design session with the whole team to search for solutions. Be open to changing the partner of the task's owner or to giving the task to another programming pair altogether.
4. If the team cannot catch up to the schedule, get the customer involved. Inform the customer of the current situation, and ask for advice. They can simplify the story, swap the story for an easier one, or drop it altogether.

2.21.2 – Release Steering

Steering a release is the most important aspect of XP (Jeffries, Anderson, and Hendrickson 2001, 158). Release steering encompasses tracking what has been accomplished, how fast stories and tasks are being completed, and how well the XP process is working overall.

To start the process, the team must know the answers to two questions: What is the release date? And, what will the team have accomplished at that date? If the team is working towards a specified release date, the XP team works to control scope in order to steer the project to the best possible product on that day (Jeffries, Anderson, and Hendrickson 2001, 158). Tracking the current status of the project will help guide the release. This is where the acceptance test (Section 2.9.2) and system metrics (Section 2.19.1) are used. Then each release consists of a list of stories that have to be done. During each release the team tracks what stories are finished and what stories still need work.

The XP process holds that programmers can easily estimate comparative difficulty (Jeffries, Anderson, and Hendrickson 2001, 158). This allows the programming staff to improve their estimate's accuracy as the project progresses. The speed at which the team implements story points should be stable throughout the project and the iterations can be steered to match this velocity. The total number of story points accomplished per iteration should be treated as a fact of nature (Jeffries, Anderson, and Hendrickson 2001,158). If the team wants to know what the product will look like in two months, they can multiply the number of story points per iteration by the number of iterations within the given time period.

This steering process is purposefully simple. The XP process does not recommend using a PERT chart or a project management dependency graph for steering (Jeffries, Anderson, and Hendrickson 2001, 159). In XP, release steering focuses on estimating difficulty, measuring performance, and then refining estimates. The overall focus of the XP process should be getting the most useful features to the clients in the shortest amount of time. During the release stage of the process, the team can only control the scope of the project (Jeffries, Anderson, and Hendrickson 2001, 159). "By deciding what to do next and what to defer, you can have a system that does what it must do, with the best features possible, within the time and resources you have" (Jeffries, Anderson, and Hendrickson 2001, 159).

CHAPTER 3 – OVERVIEW OF SOFTWARE PROJECT MANAGEMENT

> "Software project management requires leadership, social, technical, personality, and motivational skills. You need skills in all these areas to solve a problem whose definition changes over time, in an environment filled with risks and unforeseen, potentially critical, events." (Henry 2003, xiv)

This chapter will examine the basics of software project management. The purpose is to create a foundation of knowledge in software project management. This foundation will be used when this thesis examines the XP process and completes its goal in creating a management framework for XP project management.

3.1 – Introduction to the Basics of Software Project Management

Throughout the software management process a key word that defines the project manager's role is "responsibility." The manager is responsible for the all of the personnel, resources, tools, clients, and ultimately the successes or failure of the software project. A successful software project manager uses four basic elements of the project to guide it to success: people, process, tools, and measurements (Henry 2003, 1). These basic elements are critical for the success or failure of any software project. The process of managing a software project can be simplified to a four-step model based on these basic elements:

1. Estimate & Plan
2. Monitor
3. Control & Adjust
4. Complete

Each task in the software development process goes through these four phases. This chapter will examine the application of these four elements to the tasks within a software process. The author will provide depth to each of the four basic elements by adding software specific areas that software project managers need to direct their focus. Specifically, this chapter will address eight software specific responsibilities for a software project manager: project staff, software development process, risk assessment, project resources, schedule, project effort, measurements, and software deployment. Each of these software development tasks will be examined and defined by first estimating and planning, then monitoring, controlling and adjusting and finally completing them.

3.1.1 – Estimation & Planning Overview

This is arguably the most important phase of software project management. During the estimation and planning phase a desired goal or objective is set, the tasks to achieve the objective are proposed, costs are applied to each task, and a realistic plan is created to achieve them. Depending on the process and availability of project information these four steps can occur in a step-wise fashion or iteratively.

The process starts by defining the goals and objectives of the system. High-level objectives are refined to define the project's final deliverables. These objectives form the set of project requirements, one of which is a software product. The software team examines the requirements and objectives and breaks them down into the individual tasks needed to product deliverables and accomplish tasks. Then each task is examined and enriched to provide more detail. Through this refinement of tasks the team can define the quality, risk, organization, and costs of the system.

The project schedule, budget, level of quality, and team resources form the project plan (Callahan and Brooks 2004, 36).

3.1.2 – Monitoring Overview

At the start of the monitoring phase the software project has been defined and a solid plan is laid out. The monitoring phase of software project management can then be managed by gathering both quantitative data and qualitative information on a regular basis in order to understand the status of the tasks. This phase is essential for the success of the project. The monitoring phase starts by reviewing the stated plan and then checking the actual project status against the plan.

To be successful in this phase, the software project manager focuses on two key components; speed and accuracy. The faster information is gathered, the sooner the software project manager can control and adjust the projects resources. Also, the information gathered needs to accurately reflect the status of the task. This may be challenging as most software developers are very optimistic people. A common assurance on a software project or task is to have the developer state that the project is on time and on budget thinking that with just a little more effort they can get back on track.

Lastly, the monitoring phase is also used to track information for future projects. Understanding how long or how much effort a task took, in contrast to the estimates that were defined in the project plan, can greatly help future software projects (Callahan and Brooks 2004, 37).

3.1.3 – Controlling & Adjusting Overview

Controlling and adjusting a software project involves adjusting scope, schedule, quality, cost, risk, and personnel. Each of these areas are adjusted within the project manager's control of resources, time, and scope. This phase can be described as, "coordinating people and other resources to carry out the (project) plan" (Callahan and Brooks 2004, 115).

This phase can be compared to a pilot flying a plane across the country (Callahan and Brooks 2004, 116). The path the plane takes is rarely a straight line. Wind currents, obstacles, and time delays affect the path the plane takes. The pilot's job is to make adjustments so that the flight arrives on time. There is more to this process than just steering the plane. The pilot checks the gages and the weather, checks the airport for delays, considers alternate routes, and makes the trip comfortable for their passengers. Software controlling and adjusting is a similar process that encompasses many small adjustments of resources.

Each task defined during the estimation and planning phase is adjusted in terms of resources, time, and scope. Almost all of these tasks will likely need to be adjusted, re-estimated and re-planned throughout the software project. The goal for the project manager during this phase is to keep each individual task, and thus the whole project, on time and on budget. If the first two phases of the software development process went smoothly and accurately, this phase should be relatively easy. Unfortunately, software estimates and planning are extremely difficult to complete accurately.

General project management at this phase is focused on schedule control, change control, risk control, quality assurance, and quality control (Callahan and Brooks 2004, 125).

3.1.4 – Completion Overview

The completion phase of software project management involves two components; delivering the product and assessing the project. The first step of this phase can be critical for the success of the current project and possibly effect future projects. Product delivery is the last chance the manager and the software team will have to make a good impression on the client. Small details according to the development team may be huge problems for the clients. The project manager has to account for these details and manage the client's expectations. It is the project manager's job to ensure that all the product deliverables are in place and at the level of quality to satisfy the stakeholders.

After the product is released the project manager assesses the project, including the team, the process, and the management. Assessing the process the manager has the opportunity to review the strengths and weaknesses of each aspect of the project. Allowing the team to be involved in the assessment gives them the sense that they can improve on a subsequent project. The team can provide valuable insights into the process, tools, and measurements of the project (Henry 2003, 356). To make this phase possible the project manager must be consistent and accurate in all three previous phases of the development process.

3.2 – Estimate & Plan

3.2.1 – Staff—Estimate & Plan

Managing the software project staff can determine the success or failure of the project (Henry 2003, 3). The first step in this process is to estimate and plan for the staff in the project. This section will focus on four key areas of staff estimation and planning: staff size to match effort, staff culture, staff roles, and staff development.

3.2.1.1 – Staff Size to Match Effort—To properly match staff size to meet the project's needs, the project manager must first accurately predict the amount of effort needed for the project. This will be covered in more detail in section 3.2.6. Many of the effort estimation tools return a value based on staff-months. This value can be divided by the estimated schedule in months to determine the estimated staff size (Figure 10).

$$\frac{Staff\_Months}{Schedule} = Peak\_Staff$$

Figure 10 – Equation for estimating staff size (Farshad 2007).

The peak staff value creates a starting point for the project manager to adjust by using estimating tools to maximize the schedule, costs, and effort. The Rayleigh curve suggests that the number of people employed on software project builds up from a small number to a peak and then declines (Sommerville 2001, 532). If the cost or schedule changes, the project manager will have to recalculate the estimated staff size. Once the size of the staff has been estimated the next step for the manager is to understand and plan how the staff will cooperate during the project.

3.2.1.2 – Team Organization—Software development projects are increasingly becoming more and more complicated. Gone are the days where a computer genius sits in a corner and creates the world's next must-have software product. Today, software projects consist of more than one individual; they incorporate a number of individuals where each team member has specific talents and roles they bring to the project.

The previous section examined how to determine the number of staff members needed for a particular project. By estimating and planning the team organization of the project, the manager can estimate and plan for what those staff will be doing during the project. Team organization is address with three steps:

1) Understand the culture of your organization.
2) Review and understand each member of your team.
3) Match cultural and engineering roles to people.

Organizational culture sets the stage and expectations for the team and the project in general. These subtle expectations will influence how management, the team, and the project manager's ability to keep the software project on time and on budget. To asses and estimate the organizational culture, the project manager asks these questions (Henry 2003, 5):

- What is the typical lifecycle of projects in this organization?
- How is success measured and are most projects successful?
- Do the software engineers have a positive or negative view of past projects?
- What is the organizational acceptance of software development process, tools, schedules, leadership and other project factors?
- How will the organizational culture affect the people, software development process, tools and measurements gathered?

Next the project manager has to understand each team member. Each staff member potentially comes from a diverse and varying a background, it is the project manager's job to understand each staff member enough to match them to the proper role and be able to see and avoid potential conflicts. Every team member is different; the following list of questions will help the project manager assess their staff (Henry 2003, 6):

- What is their educational background?
- What is each member's software development experience?

- Is the staff from the same generation, if not what are the differences?
- What are each member's personality traits?
- List the strengths and weaknesses both professionally and personally for each team member.

The above list of questions gives the manager half of the information they need to assign roles. The second half of the information comes from knowing the standard engineering roles and responsibilities. This document will follow Henry's seven different engineering roles, defined as follows (Henry 2003, 6):

- Requirements engineer – Elicits, documents, clarifies, and maintains product requirements throughout the project.
- Lead designer – Evaluates, chooses, documents, clarifies, and maintains product design throughout the project.
- Coder – Implements the design and corrects coding errors in the product.
- Quality assurance engineer – Plans, conducts, and reports the results of product reviews, inspections, and testing.
- Customer liaison – Maintains timely communication with customers and users throughout the project.
- Tools expert – Installs, troubleshoots, upgrades, and maintains project tools.
- Other – Additional roles required by the project, such as web site designer, documentation specialist, hardware engineer, marketing, manager, and so forth.

At this stage the project manager has estimated the number of staff members, knows the organizational culture, and understands the engineering roles for the project. The next step will be assigning the roles.

3.2.1.3 – Staff Roles—In estimating and planning the staff organization, the manager has to understand both their staff members and the organization. This section looks at the roles for software project and lays out a plan for implementing them. In assigning roles the manager tries to maximize the strengths and minimize the weaknesses of their team. This process can be done in five steps (Henry 2003, 186):

1. Compile the software engineering tasks that need to be accomplished.
2. Take each task and categorize it under in each engineering role.
3. Analyze the amount of effort needed in each role and their importance.
4. Match personality traits and software engineering skills to each role.
5. Review the role assignments for manageability and suitability.

3.2.1.4 – Staff Development—Staff development should be seen as a win-win for both the team and the project manager. Improving the team's software engineering skills can increase the team member's abilities and their respect for the project manager as a leader (Henry 2003, 11). Estimating team needs, individually, personally, and professionally, does this. The project manager uses this information to create a plan to make the team better. This is done for each member on the team in a four-step process (Henry 2003, 11).

1. Make professional development part of the project's estimating and planning phase.
2. Identify long term and short term goals for each team member.
3. Ask each team members to formulate and define their goals for the project.
4. Have team members to keep a time log of their activities. This allows each person to know and understand how they spend their engineering time, identify weaknesses, and suggest areas of improvement.

3.2.2 – Software Development Process—Estimate & Plan

All software projects follow a software development process whether it is formally defined or not. It is the project manager's job to select a process that is effective for the development team, the organization, and the project. This process will lay out the what, when, and why for the development team and track both the past and future activities (Henry 2003, 25).

In the estimating and planning phase the manager does not just pick a process and demand that the team follow it; rather, a software process is selected based upon a combination of formal process specifications, project characteristics, and project team input (Henry 2003, 26). This can be done using one of three methods:

1. Select an existing process.
2. Modify an existing process to better suite your project.
3. Specify a process for the team and project.

Table 1 shows the standard software development processes. This list should be used as a starting point to further investigate each process to find the one that best fits the project and team.

Table 1 – Software development processes (Charvat 2003).

	Control of:						
Type	**Size**	**Quality**	**Time**	**Cost**	**Phase**	**Project Size**	**More Info**
Extreme Programming	N	Y	N	N	N	S,M	2
Code & Fix Approach	N	N	N	N	N	S	4
Waterfall	Y	Y	Y	Y	Y	M,L	1,3
Open Source	N	N	N	N	N	S,M	2
Spiral	Y	Y	N	N	Y	M,L	1
Synchronize and Stabilize	Y	Y	N	N	Y	M,L	
Reverse Engineering Development	Y	Y	N	N	Y	M,L	1
Structured System Analysis & Design	Y	Y	N	N	Y	M,L	1
1. High management ceremony 2. Low management ceremony 3. Classic "waterfall" sequence 4. Not suited to virtual teams							

When the project manger is estimating and planning a software process, they must first evaluate existing processes then decide if they are going to modify an existing process or define a new process. In tailoring an organizational process, use the list of standard software development processes as a base for specifying a new process. The manager should not just pick one and drop it in place. They need to find and modify a process that will work effectively for their existing team, organization, and project. To do this they should follow these steps (Henry 2003, 32):

1. Look at representative organizational projects and determine how this project is different.
2. Identify two categories of activities; first what this project needs from the existing process. Second, what the project does not need from the existing process.
3. Suggest changes to the existing process.
4. Document and distribute the proposed changes among the organizational personnel for review and feedback.
5. Define the changes and move decisively towards closure.

Following these steps will allow the project manger to intelligently change an in-place culture and process. Do not look to make major changes as organizations move slowly and changing corporate culture is difficult and time consuming. The project manager should remember that every process has its strengths and weaknesses, just try to find the best fit for your project.

The other option for estimating and planning for a software development process is specifying a process that fits the project and team. Here are five steps for specifying a process:

1. Know and document the goals and needs for software development process.
2. Specify a format.
3. Specify the process from the most abstract level to the most detailed level.
4. Add increasing detail as the specification evolves.
5. Gather and use the project team's input for the specification.

Specifying a process starts by documenting the type of process that will benefit the team. This is achieved by asking the team for input on the process and defining protocols for communication, quality and control mechanisms (Henry 2003, 34). Process format specification can take various forms. One example is a layered flow diagram, where a diagram is created starting with the highest level of abstraction and detail is added progressively to each level (Henry 2003, 34).

Once a process has been chosen the project manager must develop a plan to implement the process. First the manager has to be the process' biggest supporter. Next, they have to stick to their guns and not skip parts (Henry 2003, 37). Lastly, the manager must have a plan in place to adjust the process if needed.

3.2.3 – Risk Assessment—Estimate & Plan

Every software project involves risk and, unfortunately, estimating and planning these risks is never easy. Project risks involve events that could negatively impact the project. The precise likelihood and impact of these events are unknown. Software project risk can, be defined as, "an event, development, or state in software project that causes damage, loss, or delay" (Henry 2003, 107). The words "damage, loss, and delay" should convince every project manager to identify and evaluate risks. Risk assessment in the estimation and planning phase is accomplished with two steps; evaluate potential risks and plan for them.

The evaluation process is done in five steps (Henry 2003, 97). First, list the risks involved with the project. Brainstorm all the potential risks involved in the project and document them. After all the project risks have been documented, they need to be reviewed. During this process the potential risks are ranked by the likelihood of occurrence. Take the original list of risks and copy it to a new document. In this new document, add a numerical rank for that particular risk's likelihood next to the risk description. The rankings should go from the most likely, number one, to the next most likely, number two, and so forth.

The next step in the risk evaluation process is to rank the risks by their impact on the project. Take the original risk document and copy it, and like the last step add a number, low to high, next to each risk representing the impact of the risk on the project. The impact means how much this risk could damage, delay or cause loss to the project.

Lastly the two documents, the risk likelihood and risk impact, are combined into a third document, the final risk assessment document. This is done by adding up the numeric value of the likelihood rank and the impact rank to create a combined rank for each risk. This can be done in the original risk document or a new one. The resulting document should show four columns, the likelihood rank, impact rank, combined rank, and the risk description. Rows should be ordered from the highest combined rank to the lowest.

The final risk assessment document completes the estimation phase of risk assessment. Next, the software project manager needs to create a plan for each of these risks. Planning for each risk can take many forms. The project manager should focus first on the risks that have the highest combined rank. Here are some examples of risk assessment plans (Higuera and Haimes 1996).

- For each risk develop a contingency plan with a specified triggering event.
- Make a change to the product, project, or team to eliminate the risk.
- Accept the risk and create no plan, thus accepting the consequences.

3.2.4 – Resources—Estimate & Plan

Software projects have a wide variety of resources that keep the team moving in the right direction. All of these resources need to be gathered, updated, and maintained. The first step in this process is to estimate and plan for the project's resource needs. Standard software project management breaks a software project's resources into three categories; hardware, software, and support staff.

The software development process requires a wide range of hardware products. The effectiveness of the team could rely on the stability of the network, the web server, file servers, personal development computers, touch screens, PDA's, or any other piece of hardware. To estimate hardware, the project manger first identifies the project's hardware needs in five steps (Henry 2003, 112):

1. Brainstorm and list the current project's hardware needs.
2. For each item in the list, define what functionality and need it will address.
3. For each item, list the software that is required for that piece of hardware.
4. Identify the personnel who will be using the hardware.
5. Define at what stage of the project the team will need the piece of hardware.

If the project manager understands the breadth and scope of the project, they can create the list on their own. *Software Project Management,* by Joel Henry supplied the details and explanations for the above list and details below (Henry 2003, 113). However, it may be helpful to involve the engineers to get their prospective on the project hardware needs. Be careful to separate the team's wants, or wish list, from the actual needs of the project. The second step will help define why the team needs each particular piece of equipment and it will give a sense of its importance. Step three is a way to understand the total cost and commitment involved with a new piece of hardware. For example, if the team is developing a distributed cross-platform application, the team may need more than just one Solaris machine. They will need the latest operating system and the expertise to interact with it. Next, identify who on the team will be working with the new hardware. Only involve the team members who need to be involved. This keeps the rest of the team focused on their tasks and gives the team a sense of ownership and responsibility. Lastly, when will the team need the hardware? In estimating the costs for the project, it is common to break the estimates down to each stage. Will the team need a large amount of capital upfront or will the cost be more distributed throughout the project? This will help the project manager assess the risks and cost of the project more accurately.

At the end of the hardware estimation process the project manager will know all of the hardware requirements and when each item will be needed. This information takes the form of a plan; It will be used in the schedule and the risk assessment estimates and plans.

After the project manager and the team have defined the necessary hardware for the project they need to estimate and plan for the necessary software. Software development teams use a wide variety of software that can increase or decrease the productivity of the development team. Estimating the project's software needs is similar to the hardware estimation process. First the project the project manger identifies the project's software needs through the following five steps (Henry 2003, 115):

1) Brainstorm and list the current project's software needs.
2) For each item in the list, identify the version the project will use.
3) For each item, document which software product will need upgrades or service packs.

4) Assign a member of the team to take ownership of installs and upgrades for each piece of software.
5) Integrate upgrades into the schedule for minimal impact.

The software needs/identification list should be created in the same way as the hardware list. The project manager and/or the team assemble and brainstorm all potential software needs for the project. Remember to include every software package the team will use on the project; for example, compilers, IDE's, source control system, MS project, text editors, database management software, etcetera. Next, specify the required version. This is done to create a unified starting point for the team and avoid unnecessary conflicts later. Each piece of software may need to be upgraded during the project. Try to identify what pieces will need an upgrade and see if the manufacturer has a schedule for when they plan on releasing the upgrades. If any of the identified software packages will require an upgrade, assign a member of the team to be responsible for implementing them. Lastly, schedule the upgrades at a time that will minimize scheduled delays and the risks involved. The gathered software information is added to the schedule and risk assessment plan.

Next the software project manager has to estimate and plan for any outside support the project and team will need. This support can take the form of system administration, human resources, office supplies, documentation, third party vendor support, or other tasks (Henry 2003, 116). To estimate and plan for outside support the manager first identifies their resources and gathers commitments. This is done with a five-step process (Henry 2003, 116):

1. Brainstorm and list the current project's support needs and which group of people can fill those needs.
2. For each item in the list, identify when it would be needed.
3. List how the team will acquire the support (phone, locally, remote connection, etc.)
4. Get some form of commitment from each group.
5. Cultivate a relationship with the support group.

When listing the project's support needs look to other groups that can help the team avoid roadblocks and keep the project moving in the right direction. Use this gathered information to help create a schedule for the project. These two steps will identify a starting point for support estimation for the project. Step three forces the project manager ask how the support will be gathered. This begins the planning portion of this phase. Each support item will have a contact point or person and a list of how they will administer the needed support. Any red flags or contingent needs to acquire the support should be addressed at this time. Lastly, build a relationship with all of the groups that will provide support. "Effective support from organizational staff is the result of your gaining commitments and building positive relationships" (Henry 2003, 17).

3.2.5 - Effort—Estimate & Plan

Effort estimation and planning can be one of the most difficult tasks for the software project manager to accomplish accurately. Sommerville writes, "There is no simple way to make an accurate estimate of the effort required to develop a software system" (Sommerville 2001, 518). All the same, software project management and practical organization require effort estimates. Furthermore these estimates and plans will be used to allocate staff, identify risks, define resources, and estimate and organize the schedule.

Cost and effort estimation techniques follow two different approaches; top-down or bottom-up (Sommerville 2001, 518). For the top-down approach, the estimator starts at the system level and examines the overall functionality of the system and how underlining sub-functions provide functionality. Then, the estimator sums up the costs from a top-down viewpoint. In addition top-level activities such as integration, configuration management, documentation are also taken into account (Sommerville 2001, 518).

The bottom-up approach begins at the component level (Sommerville 2001, 518). The software product is decomposed into components and the effort required to develop them are calculated. All of the component costs are summed to create the total cost/effort estimation for the software product.

Both the top-down and bottom-up approaches decompose the software system into smaller, more easily estimated, parts. Sommervile offers a list of cost estimations techniques that use these approaches (Sommerville 2001, 519):

- Algorithmic cost Modeling—A model is developed using historical cost information that relates some software metric to the project cost. An estimate is made of that metric and the model predicts the effort required.
- Expert Judgement— Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached.
- Estimation by Analogy— This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects.
- Parkinson's Law— Parkinson's Law states that work expands to fill the time available. The cost is determined by available resources rather that by objective assessment. If the software has to be delivered in 12 months and five people are available, the effort required is estimated to be 60 person-months.
- Pricing to Win— The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customer's budget and not the software functionality.

For large systems several cost estimation techniques should be used and compared (Sommerville 2001, 519).

3.2.6 – Schedule—Estimate & Plan

In the schedule estimation and planning phase the process of creating the schedule becomes the basis of the plan. A software project schedule communicates project length, tasks, deliverables, milestones, risks, and critical events to the team and stakeholders (Henry 2003, 123). It also communicates commitments and dependencies between team members, the team and support staff, and stakeholders and the team (Henry 2003, 123). Schedule creation is a four-step process:

1) Estimate project size and effort (Henry 2003, 125).
2) Schedule Immovable milestones (Henry 2003, 142).
3) Schedule Synchronization points (Henry 2003, 145).
4) Facilitate communication (Henry 2003, 148).

Step one, estimating the project size and effort, has been examined and explained in section 3.2.1.1 and section 3.2.5. The size and effort estimates should be gathered in order to answer the question, how big is the project (Henry 2003, 128)? Use this answer to estimate how long the project will take to complete. Schedule length can then be used to establish start and end dates for the project. Now the project manager can decide when to do what (Henry 2003, 142). First define any immovable milestones and schedule them. An Immovable milestone is defined as, "events in a project schedule that occur on specific dates and cannot be rescheduled regardless of project factors or developments" (Henry 2003, 142). Immovable milestones are scheduled through a three-step process (Henry 2003, 142):

- Add the milestones to the schedule.
- Estimate the software system functionality at each milestone.
- Schedule the tasks that need to be accomplished prior to the milestone and work backward in time to the start of the project or the last milestone.

Once the immovable milestones have been integrated into the schedule the schedule as a whole should be starting to look more complete. The next step is to schedule all the synchronization points for the project. These are defined as, "points in a project schedule that require the project team to synchronize the contents of products, complete tasks, and reduce defects" (Henry 2003, 153). Inserting synchronization points into the schedule is done through a three-step process (Henry 2003, 145):

- Add synchronization points before major decisions or risk assessment.
- Only add synchronization points where testable, integratable, and substantial portions of the functionality are completed.
- Synchronization points should be used to evaluate product and task status.

Synchronization points are utilized to determine the status of the project and to verify that the team is on schedule for completion. They also help reduce latent defects (Henry 2003, 146). The last addition to the schedule is team communication. Here are some simple guidelines for scheduling team communication (Henry 2003, 148):

- Meeting should be managed so time is not wasted.
- Get standard format status reports to communicate between meetings.
- Record important communication so all the parties involved understand and are informed.

The final schedule should include the project's size and effort, immovable milestones, synchronization points and team communication to be considered complete. The schedule is now as precise as possible a plan for the project. It is important to remember that schedules are rarely accurate for long periods of time (Henry 2003, 124). Software project schedules can get out of date quickly and they need to be monitored and updated to reflect project status.

<u>3.2.7 – Measurements—Estimate & Plan</u>

Software project measurements are used to answer the multitude of product, process, and management questions that arise during a project. The answers to these questions can provide a real-time view of the project's progress, risks, time allocation, quality, and schedule (Henry 2003, 63). To maximize the effectiveness of measurements software project managers have to estimate and define what measurements will be gathered and specify the process needed to gather them.

When beginning the process of creating a plan for collecting measurements the manager has to understand what to measure and why (Henry 2003, 64). This is done through four steps (Henry 2003, 65):

1. Define the questions the project manager wants answered during the project.
2. Define the specific measurements needed to answer step one.
3. Define measurement gathering and how measurements will be stored.
4. Define how the measurements will be analyzed.

In defining the questions that the measurements will answer, the software project manager can utilize the following standard project management questions (Henry 2003, 64): What was the effort level for the project? Was the schedule meet? What was the quality of the product? These questions will be the foundation of the project's measurements. It is helpful to approach creating theses questions by asking what would be helpful to know at the end of the project (Henry 2003, 63).

Step two involves choosing the measurements that will best answer the questions created in step one. The specifications for the measurements should be divided into two separate categories: control measurements or predictor measurements. Control measurements are used to analyze the current status of the project (Henry 2003, 67). Predictor measurements help forecast the future (Henry 2003, 67).

An analogy of a muffin factory can better explain the difference between Control and predictor measurements. The current temperature of the ovens and time the muffins spend in them is an example of control measurements. Are the muffins over or under cooked? What changes to the current production line will improve the quality, speed, or cost. In turn, predictor measurements can focus on the same information but try to answer questions forecasting the future of quality, speed, and cost. For example, at the factory's current output, when will flour need to be ordered? How many muffins will be produced today, this week, and next month? How will the number of muffins produced influence the muffin line's ROI for tomorrow, next week, and next month?

The muffin factory analogy provides a starting point to examine software control and predictive measurements. Tracking the source lines of code, reused lines of code, number of subsystems, number of classes, defect count, and defect count per subsystem are examples of control measurements. These same measurements can be predictive when the future estimates are compared to the actual measurements against the current system (Henry 2003, 67). An in-depth examination of software measurements is beyond the scope of this thesis. What is important for this thesis is the process of using software measurements in the software engineering process.

Once the measurements have been defined, the next step becomes selecting and planning for their storage. This involves addressing when, how, and by whom each measurement is collected (Henry 2003, 63). This will be the measurement plan. The process of gathering the measurements needs to be part of the software process (Henry 2003, 79). The project manager needs to ensure that time for collecting the defined measurements is integrated into the schedule or process used by the project. Then assign the task of collecting measurements to the appropriate team members (Henry 2003, 79). This defines a plan and a person responsible for its implementation. Then define precisely when each measurement will be collected (Henry 2003, 79). Lastly, let all the team members know where the measurements will be stored.

The metrics gathered here are not perfect and don't represent precise measurements like the ones gathered from the hard sciences (Henry 2003, 66). Thus, software project measurements should be used as indicators and used to control and adjust the current status and future of the software project.

3.2.8 – Deployment—Estimate & Plan

Software deployment is the step after development and before maintenance. Deployment has to be developed and planned in advance to prepare for the completion of the product. This is done for two reasons; first, this is the when the teams' hard work is showcased to the clients. Even small mistake can relay to the client that your team is unskilled and incompetent. Second, some deployments can be as complicated as the software was being developed (Henry 2003, 357). Just as software development is estimated and planned so is deployment. Software projects have two types of potential deployment, a custom product or an off the shelf product. Project deployment for a custom application follows five basic steps (Henry 2003, 359):

1. Start to plan for delivery early in software project.
2. As soon as possible test the deployment process.
3. Address installation and deployment problems quickly.
4. Monitor and measure the deployment process.
5. Repeat until the project is completed.

The project manager should try to envision the results of the project and look for any risks involved in deployment. This is done to reduce the risks of the project (Henry 2003, 360). If the final deployment environment is not completely known, focus on the area of the deployment that will not change. Make these static requirements part of the deployment plan. Ask the team to create tests that will show the deployment's current status and effectiveness. At the estimation and planning stage, only steps one and two apply. These two steps require a different path and planning method for an off-the-shelf product versus a custom application. Figure 11 will show the two paths for estimation and planning for both deployment scenarios (Henry 2003, 362 and 366).

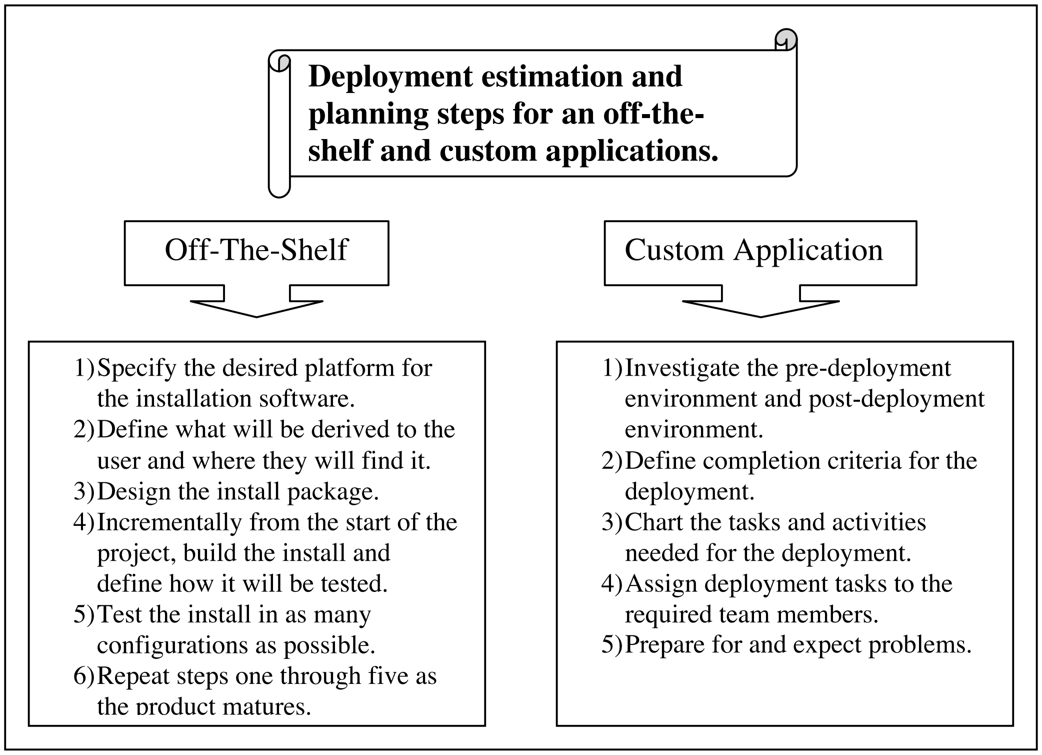

Deployment estimation and planning steps for an off-the-shelf and custom applications.
Off-The-Shelf
Custom Application
1) Specify the desired platform for the installation software.
2) Define what will be derived to the user and where they will find it.
3) Design the install package.
4) Incrementally from the start of the project, build the install and define how it will be tested.
5) Test the install in as many configurations as possible.
6) Repeat steps one through five as the product matures.
1) Investigate the pre-deployment environment and post-deployment environment.
2) Define completion criteria for the deployment.
3) Chart the tasks and activities needed for the deployment.
4) Assign deployment tasks to the required team members.
5) Prepare for and expect problems.

Figure 11 – Alternative paths for estimation and planning.

<u>3.3 – Monitor</u>

3.3.1 – Staff—Monitor

The estimating and planning phase addressed staff size, staff culture and roles, and outlined staff development. The last section gathered the necessary information to create a plan for the staff in these four areas. This section will show standard software project management techniques to identify if the plan is being followed or needs to be adjusted.

To gauge the required staff size, section 3.2.5 illustrated how to use different software effort estimation techniques. The information gathered in section 3.2.5 was used in section 3.2.1 defines how to estimate of staff size. The Rayleigh curve suggests that software project staff size will grow at the beginning and fall off at the end of any given project (Sommerville 2001, 532). From this we can recognize that the estimation and planning phase of staff size may have to be revisited and/or staff members' duties shifted. Monitoring the staff size is addressed in section 3.3.5.

Staff culture can be linked to staff conflicts and effectiveness. Staff culture should be monitored with the following three steps. First, interact with each member of the staff and assess their satisfaction with their role (Henry 2003, 264). Next, evaluate if each staff members engineering tasks are "right" for that person (Henry 2003, 264). Are they able to do the job? Last, review and update your cultural climate of the staff in its entirety (Henry 2003, 264).

A software team can have a positive culture and still not be productive. The software project manager has to monitor the staff's tasks in order to monitor their progress and output. It can be difficult to clearly define what tasks are assigned to each staff member or group. The project manager has to understand what tasks have been completed and which ones are left (Henry 2003, 267). Next, the interdependencies between tasks must also be known and monitored to maintain progress toward completion (Henry 2003, 267). Some tasks require external factors that can affect the staff. Monitor and, if necessary, adjust these factors to lessen their impact on the staff efforts. Lastly, make sure that the staff's tasks were well defined so that the monitoring process is both possible and accurate (Henry 2003, 267).

Finally, it is important to observe and gather feedback for staff improvement. Staff development was defined as part of the project's goals. Review staff development as a normal task. The project manager should monitor this process just like any of other task.

<u>3.3.2 – Software Development Process—Monitor</u>

During the monitoring phase the software development process has been defined and is currently being put into practice. The software project manager has to evaluate the software engineering tasks making sure that the chosen tools, hardware, and software are being used properly and are supporting the team (Henry 2003, 265). Monitoring the software development process starts by monitoring how software engineering practices are being carried out (Henry 2003, 265). It is necessary to review software, tools, and project hardware frequently. Monitor the implementation of the development process. Are the team members following the defined processes for requirements specification, software design, implementation, testing, quality, and overall methodology? Next, monitor every task until completion. Lastly, review the process for any problems that may have arisen.

Not all software development processes work in every project environment or organization. The software project manager has to keep an eye on avoiding process reduction, where the software engineering tasks get reduced or eliminated to a more abstract process (Henry 2003, 267). Ultimately, software development process monitoring needs to be a transparent process where the software development team knows how the project manager is monitoring the process.

<u>3.3.3 – Risk Assessment—Monitor</u>

In section 3.2.3 risks were identified and contingency plans were created. At this stage the project manager has to monitor the known risks and recognize the emergence of previously unknown risks. Team members will need to be consulted to examine their individual tasks to find risks that were not identified in the planning phase (Henry 2003, 223). Through reviewing the current project status, team member tasks, and project direction the project manager can assess and avoid risk (Henry 2003, 293). Risks assessment monitoring is accomplished in a three-step process (Henry 2003, 224):

1) Review current engineering tasks and identify new risks as the tasks progress.
2) Determine when and how risk can be detected.
3) Identify a scout to add details to emerging or unknown risks.

The monitoring phase involves gathering and interpreting information. With respect to risk assessment a scout can help in this endeavor. A scout is a staff member who researches, investigates, or explores an open issue (Henry 2003, 224). For each unknown risk, assign a scout to explore the issue (Henry 2003, 224). Create a clear and effective assignment for the scout and specify when the results are due.

3.3.4 – Resources—Monitor

Section 3.2.3 showed how a software project manager defines and plans for the project's hardware, software, tools, and support needs. This section will examine how the software project monitors these resources.

Recall from section 3.2.3 the last two steps in estimation and planning hardware are to identify who is responsible for the project's varying hardware needs and when each resource will be used. To monitor the projects hardware, be in touch with the identified hardware specialist regularly throughout the project. Check to make sure that the current hardware is meeting the project's requirements. Would different hardware make the project easier? Are their any failures or unexpected requirements that need additional hardware? These questions should be addressed at regular meetings and also cross-referenced against the planning question "When will the hardware be used." For example, if a web server was going to be brought online on a specific date follow up a day or two later to verify that the hardware is suiting the desired project needs. It best to know about problems as soon as possible, so reference the schedule to identify when the projects hardware will be used and check on its effectiveness.

Monitoring software needs are similar to monitoring hardware needs. The last two steps in section 3.2.3 instructs the project manger to identify a person responsible for installs and upgrades for each piece of software the team will be using; then integrate the upgrades into the schedule. Regularly check with the team member who is responsible for the team's software needs. Verify that the current software is meeting the team's needs and check on software upgrade status.

Finally, the project manger monitors any outside organizations or individuals who are providing support for the software project. Refer to the list of support needs from section 3.3.4. Use this document regularly to ask the team if the support was needed for each item. If the support was required, did the support personnel provide it in a timely effective manner? Was support provided as the team had planned (e.g. phone, remote access)? Was the support effective?

<u>3.3.5 – Effort—Monitor</u>

Monitoring project team size and effort can be one of the hardest tasks for the software project manager to complete. Given that team effort estimates are often inaccurate, monitoring the effort can be critical. This process is completed with three steps (Henry 2003, 278, 286, and 293):

1. Gather required information for the monitoring process.
2. Interpret the gathered information correctly.
3. Control and adjust if needed. (Section 3.4.5)

Acquiring information about the project is at the center of software project management. This process is accomplished with five steps (Henry 2003, 278):

1. Use the tasks in the schedule and make sure every task is listed.
2. As tasks are tackled, observe and discuss the process with the team.
3. Acquire information and feedback from the team on each task.
4. Use the projects planned measurements (Section 3.2.7) as a regular part of the project status meetings.
5. Interpret the gathered measurements to control and adjust the project.

During this process only monitor and track tasks that are in the schedule. The project manager cannot expect the staff to complete activities and tasks that are not scheduled (Henry 2003, 279). It is important that the project manager observe the process, not micromanage it (Henry 2003, 279). The project manager's goal is to support the team. Observing doesn't involve doing the tasks for the team members. Next, the project manager uses a variety of standard software measurements to help track and interpret the gathered information; one example could be a project scoreboard (Henry 2003, 282).

Once the needed information is gathered to monitor the project's current effort the project manager has to interpret the data. Here are six guidelines for interpreting project monitoring data (Henry 2003, 286):

- Gather information from balanced sources.
- Finalize your analysis when you have enough information and time to review.
- Do not form your analysis ahead of time and then find data that supports it.
- Actively look for other options or views.
- Choose the best view, even if it is different from your own.
- Move from analysis to making a decision in a timely fashion.

3.3.6 – Schedule—Monitor

The schedule is one of the most critical documents in any software project and it must be used everyday throughout the project (Henry 2003, 305). A software project schedule communicates the projects length, tasks, deliverables, milestones, risks, and critical events to the team and the stakeholders (Henry 2003, 123). The project manager has to fully embrace the schedule and champion its use (Henry 2003, 305). Monitoring the schedule is a hand-in-hand process with the software project manager acting as the schedule's champion. This involves talking to individual staff members about their tasks and any impact on the schedule. Asking questions and taking notes based on the schedule and task status at every meeting elevates the schedules visibility and importance. Updating and checking on the current status of tasks gives the project manager visibility on the schedule and can trigger the need to reschedule to adjust for changes in the project (Henry 2003, 306). First, all members of the software development team need to understand the schedule's importance, this is accomplished by following six guidelines (Henry 2003, 307):

- Associate all of the project's tasks, events, and issues to the schedule.
- Assess the schedule during team meetings.
- Constantly update the schedule.
- Make the schedule visible and accessible to the whole team.
- Ask for input on the schedule.
- Be the schedule's biggest supporter.

Use these six guidelines for increasing and highlighting the schedules' importance as way to monitor the schedule. These six guidelines can be compressed to two parts; first, building team confidence in the schedule and second communicating in a two-way manner about the schedule.

Associating all of the project tasks, events, and staff to the schedule is not an easy task. This process requires written communication and informal discussions (Henry 2003, 307). The project manager and the software team need to connect software events, hardware events, staff events, cultural events, and other activities that affects the schedule and need to be considered (Henry 2003, 307). Next, during the weekly team meetings the project manager should review the schedule. Team members should have a chance to comment on their tasks and how they impact the schedule.

Schedule monitoring in team meetings involves four steps (Henry 2003, 308). First, review the schedule with questions and statements. Next, ask each team member if they are on the schedule. Then, address each answer with follow up questions about possible problems. Last, restate and recap what the team stated about the schedule and identify possible risks.

At this point in the schedule monitoring process the team has embraced the importance of the schedule and it is regularly being discussed. This process has started to flush out any schedule slips and trigger necessary control and adjustment actions. It is important to note that not every event or bumpy task causes a schedule slip (Henry 2003, 310). Here are six steps a project manager can use to know if the schedule has slipped (Henry 2003, 311):

1. Identify all the tasks that have been extended or are past due.
2. Measure how much each task has slipped with respect to the schedule as a percentage of the original schedule time.
3. Find any tasks that depend on the slipped task.
4. Look for ways that the team can remain on schedule.
5. Identify a best and worst-case scenario for the schedule.
6. Use the gathered information to decide if the overall schedule has slipped.

When monitoring the schedule for slips, the manager needs to keep in mind from where they are gathering their information. Team members who are responsible for a task that has slipped may be optimistic about the size of the slip (Henry 2003, 312). During this process the project manager should be realistic about the effects missed tasks have on the schedule.

<u>3.3.7 – Measurements—Monitor</u>

In the estimating and planning section the project manager started by defining the measurements and identifying how they would be gathered, stored, and then analyzed. In the monitoring of measurements phase some of these steps will be reviewed and more detail added. It is important to know that planning to capture measurements is different from actually capturing the measurements (Henry 2003, 258). If the project manager expects the team to consistently and accurately gather measurements the team needs to understand the collection process (Henry 2003, 258).

The monitoring phase of software project measurements involves implementing the plan and verifying its success or failure. This process involves seven steps (Henry 2003, 259):

1. Assigning staff members to capture all the planned measurements.
2. Define and communicate to the staff members, how to capture the measurements.

3. Define the frequency of the measurement capture.
4. Define how to store the measurements.
5. Define the location to store the measurements.
6. Monitor and review the collection process.
7. Analyze the gathered measurements.

The monitoring phase for measurements is more active then some of the other monitoring phase tasks. This is because measurements don't always fit into the software development process (Henry 2003, 259).

3.3.8 – Deployment—Monitor

Deployment of a software product starts as early as possible in the process. Many developers will want to push deployment tasks off to the end of the project. During the monitoring process the manager has to verify that all the deployment tasks are being addressed throughout the project. Deployment tasks need to addressed in a circular process where the team deploys the product, then tests deployment, then build a new version of the product that is again deployed and tested (Henry 2003, 268). Standard task monitoring can be applied to deployment tasks (Henry 2003, 279).

- Observe and discus the deployment tasks being performed.
- Acquire information and feedback on the deployment process to assess its effectiveness and execution.
- Measure the status and execution of the deployment process.
- Use the gathered measurements for decision-making.

Throughout the deployment monitoring process the project manager tracks customer experiences (Henry 2003, 370). This can be done by email, notes, impromptu conversations, or by observing the end users. This allows the project manager to record requirements for future releases. Lastly, when observing the deployment process, it should be compared closely to the completion criteria outlined in the estimating and planning phase.

3.4 - Control & Adjust

3.4.1 – Staff—Control & Adjust

Staffing problems need to be addressed quickly. This section looks at the staff and examines ways a software project manager can control and adjust the team. Staff control and adjustment are handled in these four areas: managing staff, facilitating communication, directing the staff, and refocusing the staff.

Software engineers can be opinionated, headstrong and talented. It is the project manager's job to make sure these talented staff members are getting the project's tasks done on schedule. One of the keys to managing the staff is not to dictate to them (Henry 2003, 8). Software staff management can be conducted with six guidelines (Henry 2003, 9):

- Support the staff without micromanaging.
- Review the staff and their assigned tasks, not the people.
- Coordinate, don't manipulate.
- Solve problems with your experience, not your position.
- Remove obstacles for the staff.
- Focus on the project and the staff's needs.

In managing the staff the project manager also needs to balance the need of the project against the needs of the staff. Five guidelines for balancing project needs (Henry 2003, 93):

- Remember the trade-off between features, resources and time. As features are added, time must be increased if resources remain the same. Conversely, if features are added then resources must be increased if time is constant.
- Understand the difference between productivity and quality trade-off in staff decisions.
- When making project decisions, consider cultural versus technical trade-offs.
- Try to maintain equilibrium between meeting time and individual work time.
- Look at the trade-offs before making a final decision.

Staff communication is critical to the success of any software project. The project manager has to make sure the team members are communicating effectively and resolving problems as they arise. The following five guidelines will help the project manager facilitate communication between all the staff members (Henry 2003, 148):

- Schedule project meetings, but don't waste all the staff's time in meetings.
- Manage and focus team meetings to make them efficient and effective.
- Acquire status reports in a standard format for communication between meetings.
- Record and document important meeting notes so the team is informed.
- Meetings and communication needs to be allocated in the schedule.

Directing the staff involves tasks that are at a lower level than managing the staff. Directing involves moving the staff towards the goal of finishing the project. Listed are nine guidelines for directing the staff (Henry 2003, 251):

- For tasks and activities move quickly to add details and clarification if requested.
- Inform the staff of task priorities.
- Have the staff focus on the project goals.
- Ask questions, do not issue orders.
- The software development process, schedule, staff roles, tasks, and measurements need to be important to the staff and project manager.
- Know when a problem is serious versus trivial.
- Move quickly to solve problems.
- Completely follow through on solutions.
- Mean what you say and say what you mean.

Sometimes during a project staff members can spend time on side projects, low-level tasks, team tools or completely unrelated activities. When the project manager discovers the team thrashing on these tasks, it is time to refocus the team. Refocusing the staff can be accomplished by asking the staff member questions about the task (Henry 2003, 252). The goal here is to first understand what the staff member is doing and why. The task may be important and thus needs to be brought to the attention of the project manager. By asking questions that center around the project, project needs, goals, and risks, the project manager can adjust the staff members' tasks without being seen as an authoritarian or ego driven manager.

3.4.2 – Software Development Process—Control & Adjust

During every software development project the manager puts in place a software development process. Standard software development processes do not perfectly fit every software project. During the monitoring phase the project manager uses measurements to evaluate the effectiveness of their process. If the software development process is not as effective as it could be, the manager now has to tailor the process to fit the needs of their project. The problem of adjusting or tailoring the software development process can be approached as a general project problem. Solving software project problems is accomplished in a six-step process (Henry 2003, 297):

1. Completely understand what the problem encompasses, and how it affects the project.
2. Bring the staff into the problem solving process.
3. Define and review multiple solutions.
4. Choose the best solution for the problem.
5. Define a plan that implements the new solution. See section 3.2.2
6. Monitor the new software development process. See section 3.3.2

3.4.3 – Risk Assessment—Control & Adjust

Risk management can be a large part of a software project manager's job. Section 3.3.3 focused on monitoring and updating the risks that were identified in section 3.2.3. This section will describe how to create a backup plan for the risks that have become issues that could negatively impact the project. Once a risk has materialized the project manager has to make changes to mitigate the impact; making a backup plan does this. Creating the backup plan is a five-step process the first two steps of which were covered by sections 3.2.3 and 3.3.3. This section will examine the last three steps (Henry 2003, 224):

1. Assign a scout to examine and gather information for the identified risk.
2. Examine the risk's impact and the teams suggested actions for the risk.
3. Specify which actions to take and implement them.

3.4.4 – Resources—Control & Adjust

At this phase the project manager has defined, gathered, and monitored their project resources, the next step is getting and maintaining the support for those resources. Commonly, staff members from outside of the development team provide a software project's hardware, software, and support staff. The project manager has to maintain a good relationship with and be diplomatic when exerting, pressure to obtain results from these outside team members (Henry 2003, 234).

Controlling and adjusting the software project's resources involves a three-step process. First, use section 3.3.4 to identify when the team has a resource problem. Second, find a solution for the resource problem. Last, implement the required resource adjustment.

Finding solutions to resource problems can be abstracted to a six-step process (this process can be applied to hardware, software and support staff problems) (Henry 2003, 297):

1. Understand the resource problem and how it will affect project.
2. During the solution process, include the development team.
3. Define and evaluate multiple solutions.
4. Select the solution that best fits the problem.
5. Define a plan that executes the solution.
6. Execute and monitor the plan.

When controlling and adjusting hardware problems the manager needs to balance the costs of the hardware verses the cost of the team trying to scramble for a Band-Aid solution. Often having the programmers scramble to set up a new development server vastly outweighs the cost of purchasing the server preconfigured. In addition, most hardware purchases rarely go unused on the current project or subsequent ones. In the process of finding a solution for the hardware problem, the manager first needs to look for solutions that don't drastically differ from the eventual deployment environment, if applicable. Second, the knowledge and steps to set up and configure the hardware should be captured during the process in order to mitigate risk. Third, keep the team focused on the goal of the project and minimize the distraction of a new piece of equipment.

Software problems can be more complicated to control and adjust because of the multitude of choices of products on the market. First the team needs to completely identify the software problem. The manager has to attack the generalities of the problem and work hard to find the core of the problem. During this process the team needs to agree on the problem definition before searching for solution. Having team members work in pairs or in a group and come up with multiple solutions can minimize every team member coming up with differing software products from differing vendors. The manager may have to intervene and use their leadership skills to come to a software solution in a timely manner. Last, keep the team focused on the final goal of the project.

Support staff problems identified in section 3.3.4 should to be documented and brought to the attention of the proper personnel. The manager has to be diplomatic and focus on the problem and what the support staff can do to solve the problem. Verify that any obstacles, like firewall configurations, are removed so the support staff can provide the service they were slated for. Having outside team members providing support to the team induces risk and the manager should create a backup plan if the support staff cannot or will not provide the required services.

3.4.5 – Effort—Control & Adjust

Software development effort will rise and fall through every software project. The project manager has to maintain a high but sustainable level of effort for the team. If the level of staff effort has risen above the normal rise and fall of a software development project and the level is not sustainable, it is time to make adjustments.

As Fredrick Brooks states in the *Mythical Man Month,* "Adding manpower to a late software project makes it later" (Brooks 1995, 25). This is because the added overhead each new employee brings, specifically training time and added communication, negatively impacts the entire team. Adding additional staff is an option when the tasks can be partitioned with little communication with the rest of the team and with minimal dependence on the tasks. The main tool used for this control and adjustment should be the schedule, which will be covered in section 3.4.6.

3.4.6 – Schedule—Control & Adjust

During the schedule monitoring process the team determined if the schedule had slipped and by how much. This section will use that information to reschedule smartly (Table 2) to limit the overall damage to the project. Before we inform the customer, management, or any stakeholders, the project manager needs to make a definitive plan on how the project will proceed despite the schedule change, so they don't have to break this type of bad news over and over again (Henry 2003, 314).

The choice between using forward scheduling and backward scheduling may or may not be an option for the project manager. Forward scheduling is the most desirable, but immovable milestones or project synchronization points my force a backward scheduling technique.

Table 2 – Four steps for rescheduling smartly (Henry 2003, 314).

1) Choose to use forward or backward scheduling techniques.	
If using forward-scheduling	**If using backward-scheduling**
2) Take into account any other likely slippage, estimate the slipped tasks.	2) Use additional resources where possible.
3) While considering the inter-dependencies of all the required tasks, carefully reschedule them.	3) Reduce the scope of the software
4) Update, review, revise and document the new schedule.	

When using the forward scheduling method carefully lay out the new tasks with their new time estimates (Henry 2003, 315). Look for faults in the new schedule and ask yourself and the team if the schedule is realistic. Also take into account any holidays or vacations and review the history of the project and the team. What percentage of hit to missed milestones or synchronization points have they made?

The backward scheduling method is more constraining. The project manager‘s choices are limited because schedule time has not increased. This leaves the options of adding more resources, limiting scope, or producing a lower quality product. Earlier sections on effort have suggested that adding more people may do more harm than good. When adding resources try to leverage available resources to tasks where they require little communication with the rest of the team and the tasks are not dependent on other tasks. Next, if the project manager refuses to reduce the scope and drops tasks that ensure quality, like testing, the product quality will suffer (Henry 2003, 317). To make matters worse, reducing scope with features that were low-priority doesn't always reduce the schedule (Henry 2003, 317). Last, the new schedule has to be accurate, so take a very realistic approach to controlling and adjusting the schedule.

3.4.7 – Measurements—Control & Adjust

In controlling and adjusting measurements gathering during a software project it is more important to follow through completely and collect the measurements, than to make changes to the measurements themselves (Henry 2003, 170). If the manager feels some measurements are unnecessary they should inform the team of why the measurements were removed. If new measurements are added to the project, note when the collection of these new measurements should start and when they were identified to the team. It is important that the project manager is convinced that every measurement is needed, will be captured throughout the project, and will be used to manage and assess the project (Henry 2003, 262).

If the above criteria for capturing project measurements are not being met, the project manager must solve the measurement problem quickly. This can be done with five general steps (Henry 2003, 297):

1. Understand the measurement problem (accuracy, effectiveness, or collection issues) and understand the effect of the problem.
2. During the solution process, include the development team.
3. Define and evaluate multiple solutions.
4. Select the measurement solution that best fits the problem.
5. Define a plan (Section 3.2.7) that executes the solution.
6. Execute and monitor (Section 3.3.7) the plan.

To recap, if a measurement is unused, inaccurate, or not being collected the problem measurement needs to be addressed, by finding a solution, making a plan, and monitoring the new measurement plan.

3.4.8 – Deployment—Control & Adjust

Deployment of your software product is the most critical time for the team and the project manager. This is when the client views the team's product and effort first hand. A small mistake can put the team in a bad light. The project manager needs to be a detail oriented about the deployment tasks and risks (Henry 2003, 368). During deployment the manager should leverage the preparation, research, and testing done prior to final deployment (Henry 2003, 368). Prior to the deployment process, assign problem identification and investigation roles, and define who is responsible for solution responsibility (Henry 2003, 368). Assign the person in charge of problem identification to work with other deployment roles. If your product has multiple application areas, for example web services, database, and thick-clients, it is helpful to assign application area specialists to address any problems that arise.

During this process, the manager should avoid assigning blame but instead and focus on finding problems and solving them (Henry 2003, 368). This should be done by consulting the deployment team members, considering multiple solutions, and choosing a solution that meets the projects predefined engineering goals (Henry 2003, 370). Lastly try to avoid untested late-night fixes.

3.5 – Completion

3.5.1 – Staff—Completion

In the completion phase for the staff, the manager is bringing the projects staffing tasks to a final ending point. This phase involves three steps; staffing down, allocating staff to the assessment process, and allocating staff for support.

In this phase the project's staffing needs are in decline, now that the product has been deployed the team can be reallocated. In the staffing down process the manager works towards accomplishing two goals:

1. First, define who on the staff is not needed and provide them a pathway to be officially off the project.
2. Allocate the required staff for project assessment and provide a support staff if needed.

In examining the existing staff, the project manager starts by looking at the existing team organization (Section 3.2.1.2). Examine each role and what each staff member's current role in the project is. Ask each team member individually what tasks they are currently working on with respect to their defined roles. If they are not needed or have no tasks to complete return the staff resource to another project's resource pool. Prior to releasing a staff resource make sure they have:

- Checked in all of the project's code;

- Archived any project-related data, including email, notes, documents and other project related files;
- Removed project files and data from their individual workstations;
- Last, have each staff member officially sign-off to document that their work on the project has ended.

Using the staff members who have signed off on the project, allocate them to the project assessment process. The purpose of project assessment is to gather the lessons learned so they can be applied to future projects (Henry 2003, 380).

The first phase of the project assessment is planning for the assessment. The project manager organizes the assessment into project areas and project history. Project areas focus on major project areas (Henry 2003, 381). Using this chapter as a guide; those areas will be staff, the software development process, risk assessment, project resources, the schedule, team effort, project measurements, and deployment. The team will also examine the project history.

Using the staff estimation and planning tools from section 3.2.1 and section 3.2.5, the project manager allocates the staff to address the project assessment needs. This need will vary with the size and depth of the project assessment. As a staring point for the staff interviews, start with this list and add to it where needed (Henry 2003, 383):

- How well did the team work together?
- What staff decisions were made when, and how did they affect the project?
- What major staff problems were encountered?
- Did solutions to staff problems work?
- Was the staff comfortable in their roles?
- As the project progressed how did the culture change over time?

The answers to these questions will be analyzed in the project assessment report. These steps will be covered in the measurements section 3.5.7. Gathering the project history involves dividing the staff to review and assess seven different project areas. The project manager has to emphasize the importance of the staff to focus on what can be improved (Henry 2003, 386). With a broad-spectrum approach the staff will need to assess these areas (Henry 2003, 385):

- Requirements assessment – the creation, modification, and quality of the requirements process, the products created, and effect of the requirements over the life of the project.
- Design assessment – the creation, modification, and quality of the design process, the system architecture, and effect of the design over the life of the project.
- Code assessment – the creation, modification, and quality of the coding process, source code, and effect of changes in the code over the life of the project.
- Testing assessment – the estimating, planning, monitoring, and adjusting of the testing process, test results, and affect of the testing over the life of the project.
- Documentation assessment – the creation, modification, and quality of the documentation process, the user documents, and effect of the documentation process over the life of the project.
- Management assessment – the cost, effort, schedule, and important decisions made over the life of the project.
- History assessment – the story of the project, with overall results, failures, successes, culture and events.

As some of the staff will be allocated to the project assessment, others may be assigned to providing additional customer support. This support can be in the form of answering phone calls about the product, replying to email, or on-site fine-tuning (Henry 2003, 371). Providing customer support can be key for gaining customer satisfaction, and most software projects are not completed until some level of support is provided. Using the de-allocated staff members continued client support is planned for and implemented using these five steps (Henry 2003, 371):

1. Define the type and needed length of support in the planning stages of the project.
2. Create project support requirements.
3. Use request-assignment pairing to plan for the project's support needs.

4. Evaluate the quality and performance of the performed support.
5. Make the end-point or completion criteria highly visible.

Once the project's products have been delivered and customer support has been provided, the staff has to focus on assessing the project. Each of the next six sections will break down the project assessment into specific areas of focus. This section focused on planning the assessment, the manager's tasks and staff tasks. The subsequent completion sections will focus on creating and analyzing a project assessment report for each of the eight project domains examined in chapter three.

3.5.2 – Software Development Process—Completion

Once the software project has completed all the major tasks, all that is left is the project assessment. An important part of this assessment is the development process assessment. The project assessment report will focus on the overall success, which may be critically linked to the process assessment. The process assessment is used to make improvements to the organizational process and is applicable to multiple projects across the organization (Henry 2003, 379). This section will focus on the software development process for this project, not the overall organizational process.

There are questions about the software process that need to be answered for the assessment report. This list is a starting point and can be added to if needed (Henry 2003, 382):

- What software development process was used for this project?
- How did the project's process differ from the defined software development process?
- What parts of the process worked?
- What parts of the process need improvement?
- Did the teams tools support the process?
- Was the process effective overall?
- Was the software development process the correct one to use?
- Specifically, what improvements should be implemented in the future?

This information will be compiled, analyzed, and reported in the measurements section (3.5.7).

3.5.3 – Risk Assessment—Completion

The completion phase of risk assessment includes analyzing risk assessment and mitigation and then adding this information to the project assessment report. Listed bellow is a basic set of questions that need to be answered about the project management of risk (Henry 2003, 383-384):

- Which risks were encountered and what was their overall effect on the project?
- Gather all the projects risk assessment and mitigation efforts for the project.
- Were the mitigation efforts successful?
- Did the project encounter risks that were not accounted for in the risk assessment process?

This information will be compiled, analyzed, and reported in the measurements section (3.5.7).

3.5.4 – Resources—Completion

The completion phase for the project's resources focuses on scaling down. Once the product is delivered to the customer, the project's resources can be allocated to other organizational needs. This scaling down process can be generalized to a four-step process:

1. Freeing up resources.
2. Archiving the resource or configuration.
3. Upgrading if needed.
4. Analyzing the resource effectiveness.

The project's hardware resources have to be accounted for and released from the project. Other departments can complicate this process or organizations hosting project hardware, like web servers, database machines, demo machines, or sensor systems. These systems can be assimilated into other teams, the client's testing environment, or worse, used for the released product. Steps for scaling down hardware resources are as follows:

1. List all of the project's hardware, its location, contact person, and any other personnel who might be affected by its removal.
2. Assign a staff member or members to be responsible for each piece of equipment.
3. Contact all the personnel who may be using the hardware, inform them that the projects resources are currently being allocated to other projects, leave the contact information for the staff member responsible, and a date when the team will free up the resource.
4. Monitor and follow up on any potential problems with the scaling down process.
5. Have each staff member capture each piece of hardware configuration for documentation purposes (e.g. web server, database servers, etc.)
6. Use the time to examine each piece of hardware to see if it needs to be upgraded.
7. Analyze the project's hardware effectiveness, costs, and use in the process assessment report.

The project's software completion phase follows the same four general steps for resource completion. The freeing up of software resources can involve removing a web application from a testing, deployment, or production server. Database servers like, SQL Server or Oracle are large investments for any organization, now that the project is completed any database resources should be removed or reallocated. During the removal and freeing of software resources the team has to archive the set of software that encompasses the software project. This involves saving the versions of the database, development environment, compiler, runtime, third party tools, and any other software that was used to create the software product. During this process is a good time to examine if any of these tools need to be upgraded for then next project and to determine the gained benefit of the upgrade. Lastly, review the effectiveness of the software resources that the project used.

The completion of a software project does not always have the same impact on the support staff. The support staff can be an internal organizational department or an outside contractor. For both types of support staffs the manager has to finalize the support agreement. Are there any final bills to be paid? This can involve negotiating any billing disputes. Does the project still have billable hours left in the contract with the support staff? Do both parties agree that the support agreement has been fulfilled and completed? The project manager may or may not have to use this support staff in the future, so one of the project manager's tasks during this process is to maintain the relationship they have formed with the support staff. Next the manager has to archive the support staff contact information and contract. Lastly, review the use of the support staff throughout the project and their effectiveness.

<u>3.5.5 – Schedule—Completion</u>

The completion phase for the schedule involves two parts; administrative close out on the schedule, and adding the project's schedule data to the project assessment report.

The schedule's administrative close out focuses on archiving and documenting the schedule. Schedule changes have occurred throughout the project; this is the time when the project manager updates the final schedule to represent the actual project. The actual dates and length of schedule tasks, milestones, and synchronization points are updated to accurately represent the real project. It is important to keep the final schedule in the same format as the original schedule (Section 3.2.6) for comparison and archiving purposes. Once the final schedule is created any measurements that were being collected for the schedule should be collected and tallied from the updated schedule. The last administrative close out process is to archive all of the schedule documents so they can be retrieved and analyzed at a later date. Use the plan for collecting measurements in section 3.2.7 to archive the final schedule.

The next task in the schedule completion phase is to analyze the schedule and add the schedule data to the project assessment report. Then the staff and project manager have to answer a set of basic questions about the project schedule (Henry 2003, 384). To do this, the project manager compares the final schedule with the estimated schedule from the beginning of the project.

- Which products where started when?
- How did each product grow?
- How did each of the project's products change over time?
- Did major product rework occur, if so, when?
- How many synchronization points and milestones where hit and missed?
- How accurate was the schedule?

This information will be compiled, analyzed, and reported in the measurements section (Section 3.5.7).

3.5.6 – Effort—Completion

The completion phase for the project effort has two tasks; creating the final tally of effort for the project and adding the project's effort data to the project assessment report.

The actual effort of the project needs to be recorded at this time. Section 3.2.5 listed six different ways to estimate the effort of a software project. Using the same format and definitions of effort from the beginning of the project, analyze the final schedule and project measurements to comprise the actual effort of the project. Creating the final effort tally in the same format as the estimated effort will allow any post-project analysis to compare apples to apples. Once this information has been compiled and standardized, it needs to be archived. Use the plan for collecting measurements in section 3.2.7 to archive the final effort tally.

Now that the project's actual effort has been compiled the project manager can compare and analyze the project effort distribution. Listed below is a basic set of questions that should be added to the project assessment report with regards to the project's effort (Henry 2003, 382-383):

- What was the effort distribution between staff members per task and phase of the project?
- By week and by month what the effort for each task and phase?
- What was the project cost by phase, role and overall?

This information will be compiled, analyzed, and reported in the measurements section (3.5.7).

3.5.7 – Measurements—Completion

The completion phase has focused each of the eight subsections on finishing any additional tasks, if there were any, and compiling their subject area's measurements so they could be added in a final project assessment report. This section will examine how to handle missing information, analyze measurements, and how to interpret and present the results.

During the assessment gathering process the staff will identify missing data in the gathered measurements that would have been useful in the analysis process. Missing measurements are to be expected and this information needs to be captured so it can be used on future projects (Henry 2003, 383). Recording the missing measurements involves three steps (Henry 2003, 383): First, ask why the new measurements are needed. Second, find out where and how they could have been captured. Third, define where they need to be stored. These steps will help refine the software project measurements for the next project. Conversely, some measurements gathered for the project may not have been used, or were not effective in answering important questions. Look for ways to improve or eliminate measurements that didn't provide information that was valuable to the project.

Once the measurements have been gathered they need to be analyzed. During the process it is important to understand that a single measurement could be interpreted in more than one way (Henry 2003, 387). T o effectively analyze the gathered measurements follow these guidelines (Henry 2003, 388):

- Make use of tools for accuracy and speed.
- Analysis should be simple to perform and explain.
- Refuse to accept any preconceived conclusions.
- Think about multiple interpretations.
- Use project information.

By following the above guidelines, the measurement analysis should contain simple statistics, like mean and median, and simple views of trends over time, such as pie charts and bar graphs (Henry 2003, 389). One last note on measurement analysis; use normalized measurements in trends and for comparisons (Henry 2003, 389).

The presentation of the assessment results is the last task for the completion of the project and the completion phase of software project management. The goal of the presentation is to improve the abilities of the software project manager, the staff, and the organization (Henry 2003, 393). Follow these guidelines for presenting the assessment analysis (Henry 2003, 393):

- Define the format at the beginning.
- Link the gathered questions to the data, to analysis, and to improvements.
- Be a leader and dish it out and take it.
- Find improvements not problems.
- Remember that assessments are only valuable if the improvements get implemented and used.

3.5.8 – Deployment—Completion

The completion phase for project deployment has one critical task, obtaining client signoff. Client signoff involves having predefined completion criteria that both the team and the clients agree have been accomplished. The completion criteria should involve what functionality the users will be able to perform with the software, what administration the software will require, and any maintenance (Henry 2003, 367). Exact completion criteria for all projects will not be specified in documentation.

CHAPTER 4 – MAPPING SOFTWARE PROJECT MANAGEMENT TO XP

4.1 – Introduction

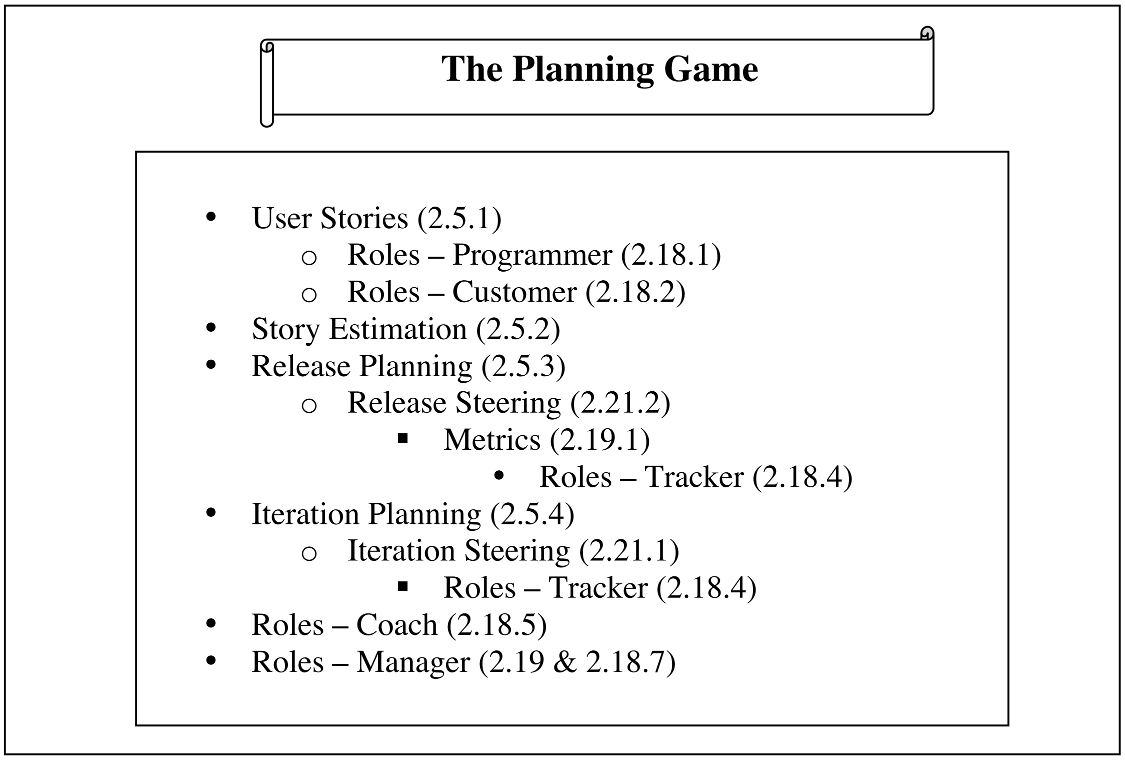
This chapter will examine how Extreme Programmings' 12 practices have addressed standard software project management practices. Chapter 2 will be used as a reference for clarification of the XP practices and Chapter 3 for standard software project management (SPM). This chapter will examine the shortcomings of each XP practice; holes where project management issues have not been addressed by the XP process (or are not applicable to the XP process). Next, it will examine what SPM tasks should be applied to each of the XP practices. Lastly, this chapter will propose additional ways to improve the XP process in addition to the added SPM tasks. The goal of applying SPM tasks, where needed, and proposing improvements is to create a management framework for XP project management that gives the XP managers the tools to utilize the XP process and still meet organizational needs for planning, monitoring, predicting, and controlling software projects.

4.2 – XP Practices

The next 12 sections map software project management to the XP processes of the planning game, small releases, system metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-hour work week, on-site customer, and coding standards. Each of these sections will be reviewed for how XP addresses the SPM processes of estimating and planning, monitoring, controlling and adjusting, and completing each practice.

4.2.1 – The Planning Game

The planning game is the most complicated and involved practice in the XP process. It involves 11 separate XP phases, which are interconnected to form the planning game practice. This section will examine how these 11 XP activities address the four phases of software project management. The diagram below (Figure 12) creates a comprehensive view of the planning game.



The Planning Game

- User Stories (2.5.1)
    - Roles – Programmer (2.18.1)
    - Roles – Customer (2.18.2)
- Story Estimation (2.5.2)
- Release Planning (2.5.3)
    - Release Steering (2.21.2)
        - Metrics (2.19.1)
            - Roles – Tracker (2.18.4)
- Iteration Planning (2.5.4)
    - Iteration Steering (2.21.1)
        - Roles – Tracker (2.18.4)
- Roles – Coach (2.18.5)
- Roles – Manager (2.19 & 2.18.7)

Figure 12 – An overview of the planning game and related sections within this thesis.

The following planning game sections will examine the planning game's practices to show how it addresses standard project management's estimating and planning, monitoring, controlling and adjusting, and completing phases. The two subsections (Sections 4.2.1.1.1 and 4.2.1.1.2) will divide the planning game by how the planning it addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.1.1 – The Planning Game—Estimate & Plan

4.2.1.1.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Software Development Process, Effort, Schedule) by the XP practice of the planning game during the estimate and planning phase. There are no SPM tasks that are not applicable to the planning game.

The planning game (Section 2.5) staff estimation starts with the XP team's available resources. This process does not match the size of the staff to the required effort (Section 3.2.1.1). Instead, estimation and planning of the staff focuses on the cost of the programming staff and the client support cost over time for the planning game process. The client and programming staff represent the fixed portion of the resources/features/time triangle (Henry 2003, 94). The velocity of story creation, estimation, release planning and iteration planning represents the variable portion of the resources/features/time triangle. This shows that the SPM task of matching staff to required effort (Section 3.2.1.1) is not applicable to the planning game. The planning game addresses staff team organization (Section 3.2.1.2) by having the manager fulfill this traditional role of adjusting the process to match organizational culture (Section 2.19) through local adoption. Roles for the programming staff are not estimated and planned in the planning game. The programming team is considered a fixed resource and the team members are not moved to different engineering roles to match their skills. All of the team members have basically the same responsibilities within the team and with the clients. Additional roles (Section 2.18) are estimated and planned by the XP coach (Section 2.18.6 and 2.19.2).

Estimating and planning of the software development process, with regards to the planning game, is handled by the project manager through local adoption (Section 2.19). Local adoption does not adequately address this SPM task, and this will be covered in section 4.2.1.1.2.

Standard SPM effort estimation and planning (Section 3.2.5) for the planning game is handled with a bottom-up approach (Section 3.2.5). Where each individual story represents a decomposed component for which the programmers estimate effort.

Estimating and planning of the schedule in the planning game has two components: the exploration phase (Section 2.5.3.1) and the commitment phase (Section 2.5.3.2). Applying SPM estimating and planning techniques (Section 3.2.6) to the exploration phase is not done; because XP focuses on the costs of available resources per time unit (weeks or months). The commitment phase (Section 2.5.3.2) of the planning game does use SPM techniques for estimating and planning the project's schedule (Section 3.2.6). However, the commitment phase in release planning (Section 2.5.3) and iteration planning (Section 2.5.4) do not fully apply section 3.2.6 methodology (Section 4.2.1.1.2). The planning game only applies section 3.2.6 methodology in a relative time chunk (iteration or release) and does not reschedule or adjust on a project wide scale.

4.2.1.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's planning game, with respect to SPM, in the estimation and planning phases and offer improvements to the process. The SPM shortcomings in this section (staff, the software development process, risk assessment, resources, schedule, measurements and deployment) are not addressed by the planning game and should be applied to the planning game process to improve it. The combination of SPM tasks and suggested improvements for estimation and planning creates the central purpose of this thesis; to improve the overall XP process.

In XP, the planning game phase for estimating and planning for the staff are missing some SPM techniques (Sections 3.2.1.3 and 3.2.1.4) that must be applied to the process. First, the planning game only partially addresses staff roles (Section 3.2.1.3). The programming staff and the client are the only roles addressed by the planning game. The planning game's tester (Section 2.18.3), tracker (Section 2.18.4), coach (Section 2.18.5) and consultants (Section 2.18.6) are not addressed in estimation and planning phase of the planning game. These roles need to be addressed through SPM (Section 3.2.1.3) for the planning game. Next, the planning game does not address any staff development (Section 3.2.1.4) thus section 3.2.1.4 should be applied to the planning game's estimation phase. Because of the critical nature of the planning game process and its overall affect on the quality, schedule, and features of the XP project, setting staff improvement goals will improve overall process efficiency. In addition, the planning game process involves tasks that typical software developers are normally not required to perform. Direct client contact, requirements gathering, and listening skills, are areas where staff development goals need to be estimated, planned, and improved (Section 3.2.1.4).

The planning game uses local adoption (Section 2.19) to modify the software development process. Local adoption is very briefly defined and can be improved by using SPM (Section 3.2.2).

Risk assessment is not addressed in the estimation and planning phase of the planning game (Section 2.5). The planning game process has many potential pitfalls that SPM risk assessment (Section 3.2.3) can help avoid. The planning game process risks include: uncontrolled feature creep, unqualified on-site client, inaccurate velocity estimates, inaccurate stories, inaccurate story estimates, rescheduling errors, failure to account for all of the project stakeholders input, and applying and communicating a realistic cost-benefit analysis to story prioritization. Standard project management (Section 3.2.3) should be applied to the planning game specifically to address the address these potential problems.

This section integrates three additional tasks to the planning game's estimating and planning phase (Figure 13). These tasks will be applied to the planning game's risk and deployment SPM components. Standard project management (Section 3.2.3) lays out a solid pathway to address estimating and planning for project risks. This section adds more detail to estimating and planning for the planning game's risks in three areas: engineering risks, release risks, and iteration risks.

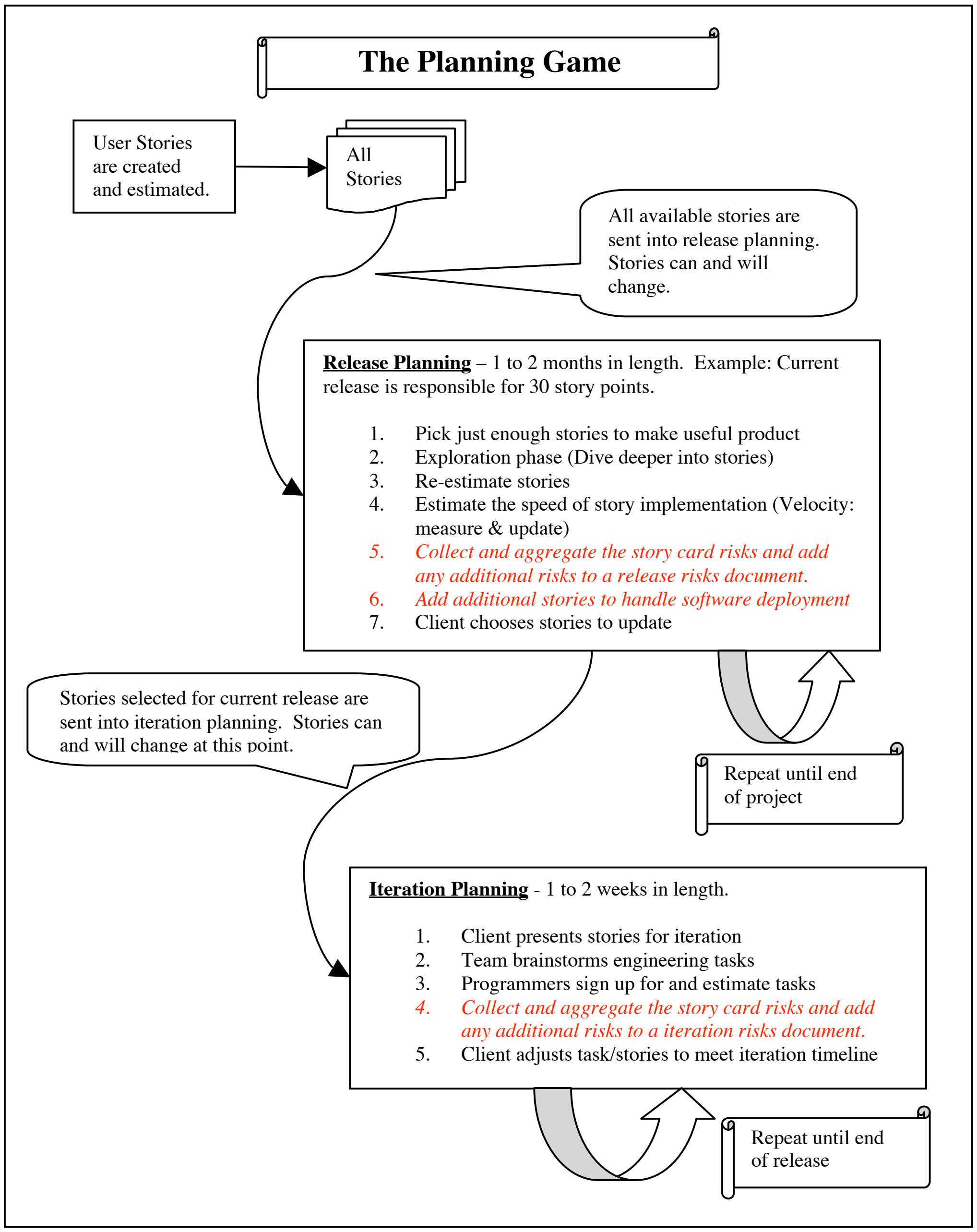

The Planning Game
User Stories are created and estimated.
All Stories
All available stories are sent into release planning. Stories can and will change.
Release Planning – 1 to 2 months in length. Example: Current release is responsible for 30 story points.
1. Pick just enough stories to make useful product
2. Exploration phase (Dive deeper into stories)
3. Re-estimate stories
4. Estimate the speed of story implementation (Velocity: measure & update)
5. Collect and aggregate the story card risks and add any additional risks to a release risks document.
6. Add additional stories to handle software deployment
7. Client chooses stories to update
Stories selected for current release are sent into iteration planning. Stories can and will change at this point.
Repeat until end of project
Iteration Planning - 1 to 2 weeks in length.
1. Client presents stories for iteration
2. Team brainstorms engineering tasks
3. Programmers sign up for and estimate tasks
4. Collect and aggregate the story card risks and add any additional risks to a iteration risks document.
5. Client adjusts task/stories to meet iteration timeline
Repeat until end of release

Figure 13 – An updated overview of The Planning Game.

Engineering tasks in the planning game are created within each story. By looking at the first task in estimating risk (Section 3.5.2), "brainstorm all the potential risks involved in the project and document them," we can apply this structure to story estimation. Figure 14 provides an overview of the updated story estimation tasks. During the story estimation process the programmers, the coach, and the client will add story card risks to each story. The XP manager outside of the story estimation process can brainstorm management risks that do not apply to each user story in accordance to SPM tasks (Section 3.2.3). The risk card will capture any technical or engineering risks associated with each story, following section 3.2.3 guidelines.

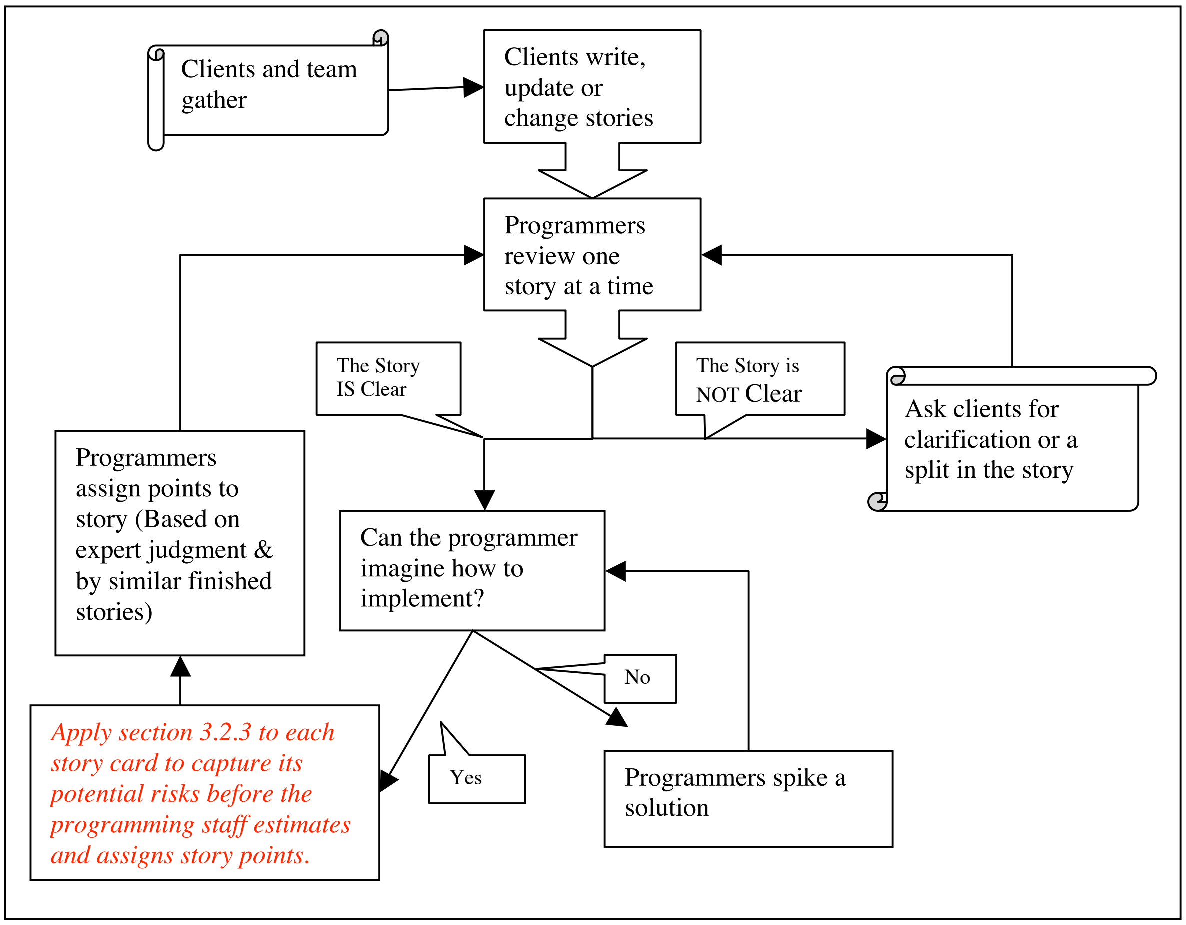

Clients and team gather
Clients write, update or change stories
Programmers review one story at a time
The Story IS Clear
The Story is NOT Clear
Ask clients for clarification or a split in the story
Programmers assign points to story (Based on expert judgment & by similar finished stories)
Can the programmer imagine how to implement?
No
Yes
Programmers spike a solution
Apply section 3.2.3 to each story card to capture its potential risks before the programming staff estimates and assigns story points.

Figure 14 – Updated story estimation.

This same process of integrating risk assessment directly in the story estimation process should be applied to iteration and release planning. For release planning in the commitment phase (Section 2.5.3.2) an additional step of estimating release risk needs to be added. The team should collect and aggregate the story card risks and add any additional risks (in accordance to section 3.2.3) to a release risks document. Using XP core values as guide, the release risk document should be placed in a visible location so all the team members will have a visual feedback throughout each release. Iteration risks should be handled in the same manner. The risk document for each iteration will be a subset of the release risk document. Iteration risk documents still need to be created because the planning game process allows for new stories to be added or subtracted. The risks for both the iteration and the release can change throughout this planning phase of the planning game so this is the natural place to update these documents.

Resource estimation and planning are not addressed in the planning game process (Section 2.5). Story cards, scheduling tools, and proper facilities need to be estimated and planned for the project. Resource planning for the planning game can be addressed by applying section 3.2.4 to the planning game process.

The previous section (4.1.1.1.1) stated that the planning game's SPM approach to estimation and planning was partially handled during the commitment phase in release planning (Section 2.5.3) and iteration planning (Section 2.5.4). Through the iterative process of release planning (Section 2.5.3) requirements are gathered, estimated, and scheduled. To further estimate and plan the schedule, the planning game iterates through the planning process (Section 2.5.4). This top down approach works for the micro schedule, but the process is missing a backwards component to update the estimations and plans of the iterations in the current release and the complete schedule for the entire project. While the planning game process is designed to handle unknown or changing requirements the schedule needs to be updated to see how changes effect immovable milestones, synchronization points, and project effort. A master schedule is not accounted for in the planning game process. After each iteration's planning session the master schedule needs to be updated using section 3.2.6 as a guide. This is to reflect the changes to the current release and changes to the project as a whole. In addition each release planning session needs to apply this same methodology to the release planning process.

The planning game process does not specifically address any estimating and planning of measurements for the process. Overall, the XP process lays out basic guidelines for gathering measurements (Section 2.19) and SPM defines how to estimate and plan measurements (Section 3.2.7). Section 3.2.7 should be applied with section 2.19.1 to estimate and plan the measurements for the planning game process.

As shown in section 4.2.1.1.2, the planning game does not account for software deployment. Each release in the XP process should be a stand-alone product that could be the final deliverable. Currently the process falls short of being able to deliver a final product with every release. Section 3.2.8 should be applied using story cards for defining any deployment tasks during the release planning (Section 2.5.3) process. These new story cards need to be accounted for when the team is estimating the team's velocity. Ideally these deployment story cards would be color coded differently to make them highly visible to the team. The idea is to force the team to think and consider tasks that are not coding centric but have an affect on the success or failure of the project.

4.2.1.2 – The Planning Game—Monitor

4.2.1.2.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Software Development Process, Resources, Schedule, and Measurements) by the XP practice of the planning game during the monitoring phase. This section will also identify those SPM tasks that are not applicable (Effort and Deployment) to the planning game during this phase.

The planning game monitoring phase does not monitor the size of the staff to match the required effort (Section 3.3.1). In section 4.2.1.1.1 the planning game staff focuses on the cost of the planning game team members overtime, not on estimating and planning staff size. Monitoring the staff size and effort is not applicable because these SPM tasks were not estimated and planned in a traditional way. The monitoring of the staff's assigned tasks (Section 3.3.1) in the planning game process is accomplished by a combination of assigned roles. The tracker (Section 2.18.4) is responsible for gathering all the required measurements for the project and the XP coach (Section 2.18.5) is assigned to monitor the process.

Monitoring of the software development process (Section 3.3.2) for the planning game is not directly addressed. The XP coach (Section 2.18.5) is assigned to monitor the process, but the XP manager is assigned to make local adjustments to the process (Section 2.18.7). The monitoring of the planning game development process is partially addressed and the next section will apply SPM techniques to this process in order to fully address this problem. The monitoring of the planning game resources (Section 3.3.4) is covered by combined tasks addressed by the tracker (Section 2.18.4) and the coach (Section 2.18.5).

The planning game process handles the monitoring of the project's effort (Section 3.3.5). The monitoring of the planning game's effort is addressed by a combination of the tracker (Section 2.18.4), the coach (Section 2.18.5), and the customer (Section 2.18.2) in the planning game process. Iteration and release planning attempt to further simplify the monitoring process by decomposing the software project into smaller and smaller parts and only monitoring the current iteration and release. Unfortunately, the planning game process does not address project-wide monitoring of effort.

Monitoring of the schedule (Section 3.3.6) for the planning game process is covered by the iteration and release steering process (Section 2.21.1 & section 2.21.2). The XP team's tracker handles the monitoring of the project's measurements (Section 3.3.7) for the planning game (Section 2.18.4). Monitoring of the deployment process for the planning game is not applicable to the planning game process. The monitoring, controlling and adjusting, and completion of an XP project are handled by the practice of small releases (Section 2.6).

4.2.1.2.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's planning game, with respect to SPM, in the monitoring phase and offer improvements to the process. The SPM shortcomings in this section (Staff, Software Development Process, Risk Assessment, Resources) are not addressed by the planning game and should be applied to the planning game process to improve it.

Surprisingly the first SPM task not covered by the monitoring process in the planning game (Section 2.5) deals with the project's staff. The planning game process does not address how to monitor the staff culture (Section 3.3.1). The failure to address staff culture during the planning game is surprising because the planning game process relies on group interactions to function effectively. Section 3.3.1 should be applied to the programming team, the tracker, the coach, the clients, and the manager. For the planning game and the other 11 XP practices, applying SPM techniques to the staff is an area where the XP process lacks focus and needs improvement. In the planning game, estimate and plan section (4.2.1.1), staff development was not addressed by the planning game process. The estimating and planning for staff development (Section 3.2.1.4) is not addressed by the planning game process and should be applied to the process. With that addition, during the monitoring phase of the planning game, the XP manager has to monitor its process. Section 3.3.1 is not address by the planning game process and should be applied to monitor the staff development in this phase of the planning game.

The previous section (Section 4.2.1.2.1) showed that monitoring of the planning game development process is partially addressed by the planning game process. This is an area where the planning game did not address SPM. SPM for the monitoring of the software development process (Section 3.3.2) should be applied to the planning game process.

In examining the estimating and planning phase of the planning game this paper stated (Section 4.2.1.1.2) that many potential pit falls of planning should be addressed by risk assessment (Section 3.2.3). The lack of risk assessment in the planning game estimation and planning phase was covered by SPM section 3.2.3. The monitoring phase of the planning game also lacks any SPM techniques to address the monitoring of the risk assessment process for this phase. Section 3.3.3 should be applied to the planning game monitoring phase to address this SPM need.

The estimate and planning phase of the planning game (Section 4.2.1.1.1) asserted that the planning game did not address the processes resources. That section applied section 3.2.4 to meet this need. In the monitoring phase section 3.3.4 is not addressed by the planning game and should be applied to the process.

Monitoring phase planning game of the XP process does not address the need to monitor the deployment process (Section 3.3.8). Section 3.4.8 should be applied to the planning game process so that any unidentified tasks, or changes in the deployment plan can be updated in the schedule.

The addition of SPM to the planning game monitoring process does a good job of covering section 3.3 tasks as applicable to the planning game process. The author suggests 11 specific questions to ask the team during the planning game monitoring process:

- Are the clients creating enough stories?
- Do the stories have enough detail?
- Is the programming staff accurately predicting story effort?
- Is the programming staff effectively communicating with the clients, for stories, risks, deployment needs, and priorities?
- Is the coach effectively running the planning game process?
- Are measurements being captured accurately?
- Is feature creep affecting the overall schedule (Is the schedule acceptable to the client)?
- Is the staff (programmers & clients) improving on their ability to execute the planning game process?
- What areas could the staff improve (strengths/weaknesses)?
- How effective is current staff culture?
- How well is the planning game parts (user stories, story estimation, release planning, iteration planning) being executed (needs improvement, acceptable, good)?
- Based on the number and amount of time spent on spike solutions, does the programming staff have the technical ability to tackle the defined user stories?

4.2.1.3 – The Planning Game—Control & Adjust

4.2.1.3.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Staff, Schedule, and Measurements) by the XP practice of the planning game during the control and adjust phase. This section will also identify those SPM tasks that are not applicable (Effort) to the planning game during this phase.

The planning game process addresses many of the SPM techniques for controlling and adjusting the staff (Section 3.4.1). The managing of the staff is handled by a layering of responsibility instead of a single manager. These layers are in place to manage the staff, facilitate communication, direct the team, and refocus the staff during the control and adjustment phase. The layers of shared responsibility start with the programmers working together to solve teams and/or projects problems (Section 2.18.1). Next, the tracker serves as the conscience of the staff for their tasks and responsibilities (Section 2.18.4). Then the coach (Section 2.18.5) runs the planning game process and directs the staff. Lastly, the manager (Section 2.18.7) solves any problems that haven't been handled by the lower levels and remove obstacles for the staff. When the staff needs to adjust the course of the planning game process, iteration steering (Section 2.21.1) and release steering (Section 2.21.2) are used.

The next SPM task that the planning game addresses during the controlling and adjusting phase (Section 3.4.6) is the schedule. The planning game only partially addresses the need to control and adjust the schedule. The planning game addresses the schedule on a micro scale while never addressing the project timeline as a whole, only addressing the iteration and releases. This adjustment of the schedule is done through iteration steering (Section 2.21.1) and release steering (Section 2.21.2).

Measurements are the next SPM task that the planning game addresses during the controlling and adjusting phase (Section 3.4.7). The team's tracker has the responsibility of addressing SPM controlling and adjusting of the measurements (Section 2.18.4).

4.2.1.3.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's planning game, with respect to SPM, in the control and adjust phase and offer improvements to the process. The SPM shortcomings in this section (Software development process, Risk assessment, Resources, Deployment) are not addressed by the planning game and should be applied to the process to improve it.

In addressing the staff, the planning game process relies on the team to plan, identify, tackle, and complete most problems. The planning game process does not allow the manager to act or define what to do when the team is not progressing. To control and adjust the staff in this critical phase the, XP manager need to adjust their style of management from a traditional manager to a leader. The process has taken care of the traditional tasks (estimating and planning, monitoring, adjusting, and completion); when controlling and adjusting the staff, the manager has to sell the vision of the product, challenge the staff, enable them to tackle unforeseen problems, and focus on the end goal. The manager has to see the forest as a collection of trees and keep the team focused on the end goal. Internal conflicts and personnel issues can destroy team chemistry, so the manger has to mix the staff and their duties to focus all of their skills on problems facing the planning game's tasks and activities.

Unfortunately the planning game does not address or lay out any framework to control and adjust the software development process. The only mention of software development control and adjustment comes in section 2.21 in the management strategy under local adoption. Controlling and adjusting of the software development process needs to be addressed with SPM techniques defined in section 3.4.2.

Risk assessment (Section 3.4.3) is not covered by any of the planning game practices during this phase. The planning game tries to address development/story card risks through spike solutions (Section 2.5.2), but it fails to address potential risks from hardware, software, the staff, office politics, or other general risks. With the addition of sections 4.2.1.2.2 and 4.2.1.2.3 improvements to the planning game risk assessment, section 3.4.5 should also be added to the planning game process.

Control and adjustment of the planning games resources (Section 3.4.4) are not addressed by the practice. During this phase (Section 2.5) of the XP process the project's plan, schedule, requirements, and tasks are created. This is the time to actively manage the team's resources using the SPM techniques defined in section 3.4.4. Resources in the planning game and the XP process are largely static. The process tracks how fast the team is working and helps the clients adjust scope. The planning game process just focuses on the consumption rate of resources and typically does not address adding resources. This is a utopian idea that could never really be implemented in a realistic project. The XP manger can look for resources that could support the planning game process without changing it. Ideally, adding supporting resources takes tasks and process overhead away from the clients and programmers allowing them to focus on the process. For example, story creation speed and accuracy could be improved with the addition of a technical writer; an outside technical consultant could be brought in to help with spike solutions. Communication accuracy and speed could be improved by sending the staff to a one or two day seminar on professional communication. These suggestions do not fundamentally change the planning game process rather; they give the XP manger the tools needed to keep in operating smoothly.

The planning game's iteration and release steering process does not address SPM deployment (Section 3.4.8) during the controlling and adjusting phase. The addition of deployment controlling and adjustment to the planning game is critical to the success of an XP project. Section 3.4.8 should be applied to the planning game to address this need. Because the planning game controls the tasks and schedule for the XP project, if deployment issues are not brought up during the planning game process they will be unaccounted for at the end of the project. Deployment problems cannot be left to the need of each iteration or release. It is important for the XP manager to adjust quickly when then monitoring process exposes a problem. First, if deployment is unclear, not defined, or not functioning, the programming staff should tackle and address any deployment problems quickly with the highest priority. This is done because deployment problems almost always lead to a better understanding of the end goal of the project and tend to solve unknown requirements. First the manager should focus the programming staff on the problem by guiding the team toward a solution. Second, the manager should focus the team on cherry picking wisely; decomposing the problem and focusing the team on the easiest problems to solve that have the greatest payoff towards an overall solution do this. Do not allow the team to get stuck on individual problems that stump or stop the process.

4.2.1.4 – The Planning Game—Completion

4.2.1.4.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Deployment) by the XP practice of the planning game during the completion phase. There are no SPM tasks that are not applicable to the planning game during this phase. In the XP process the completion phase is largely overlooked. Very few of the standard project management practices for the completion phase are addressed during the planning game process.

In examining the planning game process, XP's strengths are in the combination of monitoring and controlling and adjusting. The planning and completion activities are largely ignored. The planning game only partially addresses SPM completion of the deployment phase (Section 3.5.8). Half of the deployment completion is addressed by release planning (Section 2.5.3) and by defining the requirements of the final release so the XP team has completion criteria to work towards. Having the clients sign-off on the final deliverable will be addressed in section 4.2.5, XP testing.

4.2.1.4.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's planning game, with respect to SPM, in the completion phase and offer improvements to the process. The SPM shortcomings in this section (Staff, Software Development Process, Risk Assessment, Resources, Schedule, Effort, Measurements) are not addressed by the planning game and should be applied to the process to improve it.

In the previous section, outlining the SPM tasks for the planning game, identifies a wide range of SPM omissions. XP purists might point to the fact that the planning game and XP as a whole is a document-light process and the addition of the SPM tasks add too much overhead to the agile process. This paper strongly disputes this point. The planning game uses current status and measurements to determine velocity and past experience to predict the future. Without finishing and assessing the effectiveness of the planning game process, future projects will have inaccurate or no historical context to use. Beck talks about how to "play to win," but then does not lay out a plan to win or to finish any of the 12 practices (Beck 2000, 38).

Completion of the planning game with regards to the staff (Section 3.5.1) is not addressed. The staffing down (Section 3.5.1) process in not mentioned or addressed by the planning game. The planning game allocates time for running the XP process, but it does not address any staff assessment (Section 3.5.1). The planning game does not address the need for allocating project staff support (Section 3.5.1). The planning game approaches a software development like a two speed car that has stop and go. The car is either not going anywhere or is driving at full throttle. The Rayleigh curve suggests that software development projects have a variable effort distribution over time (Sommerville 2001, 532). Applying a completion phase to the planning game's staff process would make the process as a whole more acceptable to the corporate world.

Standard project management practices define who, when, and how for the staff, in the completion phase (Figure 15). In a corporate environment the project manager has to add the why. Variable scope contracts must end and other project may need additional staffing support. Providing a 'what's next' plan for the team can better inform and motivate the team to complete the planning game process. Section 3.5.1 explains how the project staffing down process works; the combination of sections 3.5.1 and 3.1.2 should be used to create a transition plan from project to project.

The goal is to have an end point the staff knows about and is working towards. During this dynamic completion phase the XP staff needs to see, in a common area, the current staff assignments. This can be done via a simple white board or in outlook. The information must be accessible to the entire team and displayed in a manner where the team will know the assignments on a day-to-day basis.

Throughout the planning game process this paper has noted that the software development process could be altered or adapted to fit the local environment (Section 2.19). This adaptation is very loosely defined and the planning game process has no defined way to analyze its effectiveness. The planning game process does not address the completion of the software development process (Section 3.5.2). In addition, section 3.5.2 needs to be applied to the planning game to have a historical context to view past projects.

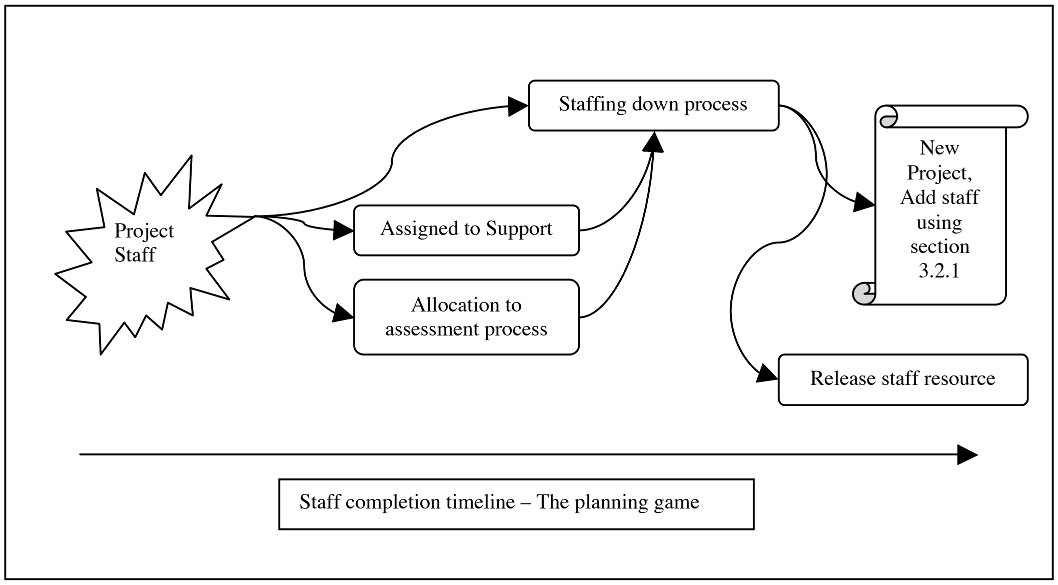

Staffing down process
Project Staff
Assigned to Support
Allocation to assessment process
New Project, Add staff using section 3.2.1
Release staff resource
Staff completion timeline – The planning game

Figure 15 – The planning game staff completion.

The planning game does not address any of the risks that were encountered, circumvented, or the ones that never materialized. The SPM task of risk completion (Section 3.5.3) is not addressed by the planning game practice and should be added.

The planning game does not cover the scaling down of and the assessment of project resources (Section 3.5.4). The addition of resource completion (Section 3.5.4) will address the SPM need for resource reallocation and provide a historical prospective that can be used to improve and adjust the planning game process in future projects.

The planning game does not address SPM schedule tasks for this phase. Neither the administrative closeout (Section 3.5.5) nor the final schedule assessment (Section 3.5.5) are covered by this XP process. This is alarming since much of the planning game's process (story estimation section 2.5.2, release planning section 2.5.3, and iteration planning section 2.5.4) rely on expert judgment and historical context to define scope, schedule and effort. Adding section 3.5.5 will improve the planning game process' overall effectiveness. The planning game's effort is related to the schedule by the way the XP process uses expert judgment (Section3.2.5). The planning game process also does not address the SPM need to review and tally the project's overall effort (Section 3.5.6).

The planning game does collect measurements (Section 2.19.1) and talks about using the collected measurements for story estimation (Section 2.5.2), but it does not address any SPM measurement completion (Section 3.5.7) tasks. Feedback is one of the core values of the XP process and obtaining accurate, effective, measurements in a standard way will help future projects.

4.2.2 – Small Releases

This section will examine how the XP practice of small releases handles standard software project management tasks and techniques. The practice of small releases in the planning game is used to increase communication, feedback, and deliver value to the customer early in the project (Section 2.6). The next four sections will examine how XP handles the practice of small releases throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases of software project management. Further subsections will identify how the small releases practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.2.1 – Small Releases—Estimate & Plan

4.2.2.1.1 – Software Project Management Addressed—No SPM tasks are addressed by the XP practice of small releases during the estimate and planning phase. This section will identify those SPM tasks that are not applicable (Staff Estimation, Software Development Process, Schedule, Deployment) to small releases during this phase.

The XP practice of small releases (Section 2.6) addresses none of the estimation and planning software project management practices (Section 3.2) for the estimating and planning phase. Furthermore, the staff estimation and software development process are not applicable to the practice of small releases. The XP process focuses on the effort expended over time of staff resources for estimating the staff available effort in the future. This process fixes the staff throughout the practice, thus estimating and planning staff size (Section 3.2.1.1) is not applicable to this practice. Next, the SPM techniques for the schedule are not applicable because the planning game practice is the central location where the schedule is created and updated. Lastly, the software development process (Section 3.2.2) is not applicable to small releases, and is handled throught this papers suggested improvements in section 4.2.1.1.1.

Deployment estimating and planning is extremely important for the small release practice, but this SPM technique is addressed in the estimating and planning phase of the planning game (Section 4.2.1.1).

4.2.2.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's small releases, with respect to SPM, in the estimate and planning phase and offer improvements to the process. The SPM shortcomings in this section (Staff, Risk Assessment, Resources, Effort, Measurements, and Deployment) are not addressed by small releases and should be applied to the process to improve it.

The first SPM task that is not addressed by small releases is the estimation and planning of the staff. Section 4.2.2.1.1 showed that the estimation of the staff (Section 3.2.1.1) was not applicable to the process, but the rest of the Section 3.2.1 can be applied to this XP practice. The execution and implementation of the small release will take staff member, time, expertise, and roles. Section 3.2.1.2 – 3.2.1.4 should be applied the practice of small releases to help the practice:

1. Organize the staff's roles for each release schedule for the project.
2. To ensure that each staff member has the same deployment knowledge and has an opportunity to release the product.
3. Ensure each staff member gains and expands on their professional development with this XP practice.

The XP process does not cover risk assessment during the estimation and planning phase of the small releases practice. Changes to the planning game's estimating and planning phase (Section 4.2.1.1.2) introduced risk assessment into that process, where the releases are originally addressed, but did not cover the specifics risks associated with releasing the product. Each release will involve different personnel, dates, and software functionality. To address these potential risks, Section 3.2.3 should be applied to the small release practice.

In the estimation and planning phase of the small releases practice, the XP process does not address potential resource needs. Section 3.2.4 should be applied the estimation and planning phase to address this need. In the planning and estimation phase a checklist of required resources for the release process should be created. Because the product will change throughout each release it will be important to maintain the resource checklist for each release. In addition different programmers will perform or assist in the small release process throughout the project, this is a way to capture and share the processes resources among staff members. Specifically the checklist should include:

- Physical location for each release.
    - Deployment type, remote, on-site.
    - Contact and availability of client personnel assisting in the deployment.
- List of all hardware that is involved (Sever, Desktops, PDA's, etc.)
- Required logins, passwords
- Support person for each piece of hardware. (e.g. Network Administrator) if needed.

The required effort to complete and implement each small release is not accounted for in the small release practice. In applying SPM tasks to the planning game's estimation and planning phase, for each release deployment tasks were integrated into the process (Section 4.2.1.1.2). With this addition to the planning game process, the required effort is crucial to its success of both practices. This addition identifies tasks that were unaccounted and thus left out of SPM tasks. Section 3.2.5 should be applied to this phase of the small release practice.

The small release practice does not have any defined measurements. Section 3.2.7 should be applied to the practice. This SPM task should be applied during the planning game's release planning phase.

4.2.2.2 – Small Releases—Monitor

4.2.2.2.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of small releases during the monitoring phase. This section will identify those SPM tasks that are not applicable (Staff Estimation, Software Development Process, and Schedule) to small releases during this phase.

The small release practice has very little substance to its implementation. Subsequently, none of the SPM techniques are applied to this practice during the monitoring phase. Furthermore, the monitoring of the staff estimations (Section 3.3.1) and monitoring of the software development process (Section 3.3.2.) are not applicable to this practice. The monitoring of the schedule (Section 3.3.6) is addressed by the planning game (Section 4.2.1.2) and is not applicable to the small release practice.

4.2.2.2.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's small releases, with respect to SPM, in the monitoring phase and offer improvements to the process. The SPM shortcomings in this section (Staff, Risk Assessment, Resources, Measurement, and Deployment) are not addressed by small releases and should be applied to the process to improve it

Section 3.3.1, monitoring of the staff, is not covered by the small release practice (Section 2.6). Given the interaction between the implementation staff, the clients testing, and guidance from the XP tester, applying section 3.3.1 would improve the small release practice.

Risk assessment monitoring (Section 3.3.3) is not covered by the small release practice (Section 2.6). The addition risk estimation (Section 4.2.2.1.2) to the small release practice, risk monitoring (Section 3.3.3) should be applied to small release practice.

Resource monitoring (Section 3.3.4) for the small release practice (Section 2.6) is not covered. The addition of resource estimation and planning to the small release practice (Section 4.2.2.1.2) compels the monitoring phase to add the SPM task of resource monitoring (Section 3.3.4) to the small releases monitoring phase.

Measurement monitoring (Section 3.3.7) is not addressed by the small release practice during this phase. Any added measurement in the estimation and planning phase need to be monitored using SPM defined in Section3.3.7. The additions to the estimation and planning phase with respect to deployment are not monitored by the small release practice. Deployment monitoring (Section 3.3.8) should be applied to the small release practice.

The monitoring phase of small releases should also follow up on the deployment/release planning added to the planning game practice (Section 4.2.1.1.2). Section 3.3.8 should be applied to the added deployment tasks in the planning game process what were addressed in section 4.2.1.1.2. Section 3.2.8 should be applied with these new tasks in mind.

<u>4.2.2.3 – Small Releases—Control & Adjust</u>

4.2.2.3.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of small releases during the control and adjust phase. This section will identify those SPM tasks that are not applicable (Software Development Process and Schedule) to small releases during this phase.

Due to lack of execution instructions, the XP process provides no SPM tasks for this phase of the small releases practice. The small release practice in not the appropriate place in which to adjust the software development process (Section 3.4.2). The schedule for the release is handled by the planning game practice (Section 4.2.1).

4.2.2.3.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's small releases, with respect to SPM, in the control and adjust phase and offer improvements to the process. The SPM shortcomings in this section (Staff, Risk Assessment, Resources, Effort, Measurements, and

Deployment) are not addressed by small releases and should be applied to the process to improve it.

The staff controlling and adjusting (Section 3.4.1) is not handled by the small release practice. The XP staff, clients, tester, assigned release implementers, should apply Section 3.4.1 to control and adjust the staff. XP does not allocate for a signification amount of time for the small release practice, so any staff issues should be handled quickly.

In the application of risk assessment (Section 3.4.3) should implemented in respect to the addition to the planning game (Section 4.2.1.1.3) for story risks. This will be important to add to the small release practice. This effectively takes the small release risk assessment and turns it into a task that the planning game can use its SPM process to monitor, adjust and complete. The upcoming testing Section (4.2.5) will add the need to define how to measure how “releases can and will be evaluated by the client.” Section 3.4.7, controlling and adjusting of SPM measurements, should be applied to the small release practice specifically to track and understand the current release and evaluate its success.

Release controlling and adjusting of resources associated with each small release in not accounted for in this practice, and section 3.4.3 should be applied to this practice. The small releases process has not accounted for any controlling and adjusting of measurements (Section 3.4.7). Lastly, deployment controlling and adjusting (Section 3.4.8) in not covered by the small release practice. Section 3.4.8 should be applied to the small release practice in this phase.

4.2.2.4 – Small Releases—Completion

4.2.2.4.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of small releases during the completion phase. This section will identify those SPM tasks that are not applicable (Staff, Software Development Process, Risk Assessment, Schedule, and Effort) to small releases during the completion phase.

During each small release the XP team does not evaluate or report on the success or failure of the project. Section 3.5.1 allocating staff to the assessment process is not applicable to the small release practice. The small release practice is used as a quick milestone, where the majority of the staff (Programmers, Clients) proceeds to the next release as quickly as possible. The staff allocation to the assessment process will be examined in section 4.2.5, XP Testing. Completing of the software development process does not handle and is not applicable to the small release practice. The updated planning game (Section 4.2.1) covers the software development process (Section 3.5.2). Completing the small release risk assessment (Section 3.5.3) will be address in section 4.2.5 and is not applicable to the small release practice. The completion of the schedule (Section 3.5.5) and the small release effort (Section 3.5.6) are handled by the planning game practice (Section 4.2.1) and are not applicable to the small release practice.

4.2.2.4.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's small releases, with respect to SPM, in the completion phase and offer improvements to the process. The SPM shortcomings in this section (Staff –Staffing down, Resources, Deployment) are not addressed by small releases and should be applied to the process to improve it.

In the completion phase of the small release practice some SPM techniques are missing and applicable to the practice. First, staffing down (Section 3.5.1) is applicable to the programming staff members who have been assigned to the process. This SPM task will bring a programming resource back in the team and also archive and milestone the release. In addition, the release's consumed resources can use SPM completion techniques (Section 3.5.4) to archive for XP projects. Lastly the SPM deployment completion (Section 3.5.8) can be applied to the small release practice. This SPM task in addition to section 4.2.5 completion phase covers any additional SPM tasks that the small release practice is lacking.

4.2.3 – System Metaphor

This section will examine how the XP practice of using a system metaphor addresses standard software project management tasks and techniques. The next four sections will examine how XP address the practice of the system metaphor throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases of software project management. Further subsections will identify how the system metaphor practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.3.1 – System Metaphor—Estimate & Plan

4.2.3.1.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of the system metaphor during the estimate and planning phase. This section will identify those SPM tasks that are not applicable (Software Development Process, Risk Assessment, Resources, Effort, Measurements, and Deployment) to the system metaphor during this phase.

The system metaphor in the estimate and planning phase is completely undefined and uses none of the standard project management practices. Most of the SPM tasks are not applicable to the process or covered by other practices.

4.2.3.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's system metaphor, with respect to SPM, in the estimate and planning phase and offer improvements to the process. The SPM shortcomings in this section (Staff, Schedule) are not addressed by the system metaphor and should be applied to the process to improve it.

Estimating and planning for the staff and the schedule are shortcomings of the system metaphor practices, but as standard project management practices (Sections 3.2.1 & 3.2.6) they cannot be directly applied to the system metaphor practice. The system metaphor process requires a lightweight process to fit into the XP values.

The system metaphor practice, along with other XP practices, was created in a vacuum, where the practice was defined, but the process did not account for its implementation. To improve the estimating and planning phase of the system metaphor we will have to add to the XP structures that control the schedule. The planning game process was set up to continually adjust and adapt to any changes that the project encountered. Section 4.2.1 added to the planning game process, as a result the system metaphor will change its structure.

The system metaphor is an XP practice that is addressed at the beginning of a project, specifically in the estimating and planning phase. To account for this time and effort in the schedule the author is adding the idea of the "first iteration."

The first iteration in the planning game is comprised of task stories (Section 4.2.1). The standard iteration planning session is comprised of these four steps:

1. Client presents stories for iteration.
2. Team brainstorms engineering tasks.
3. Programmers sign up for and estimate tasks.
4. Client adjusts task/stories to meet iteration timeline.

Where the purpose of the first iteration is to address and complete needed project management tasks instead of estimating and prioritizing engineering tasks. The project management tasks have already been defined by the XP practices, this new iteration will provide a place in the structure for these predefined practices. The first iteration will be comprised of four steps:

1. Present the management tasks to the whole XP team.
2. Define each management tasks goal.
3. Brainstorm solutions for each task.
4. Archive each selected solution.

With the first iteration built in into the planning game structure. The system metaphor can be added to the list of management tasks that need to be addressed. One additional purpose of the "first iteration" is to give the clients and the team some experience working through an iteration planning session. To closely follow the iteration planning session the management task should be comprised on story cards. This management task cards need three lines, the tasks name, its goal, and space for the team solution. For example, the system metaphor card in figure 16.

Name: System Metaphor
Goal: To create a shared story of how the whole system works

Solution:

Figure 16 – Sample system metaphor card.

Appling this structure of the first iteration and system metaphor management task, now in the estimating and planning phase, SPM staff and schedule (Sections 3.2.1 and 3.2.6) can be applied through the planning game. This method takes a XP practice and changes it into a story. This allows the effort to be constant but the staff and schedule to change.

4.2.3.2 – System Metaphor—Monitor

4.2.3.2.1 – Software Project Management Addressed—The XP practice of the system metaphor includes none of the SPM techniques during it monitoring phase. The system metaphor practice is very simple in its concept and implementation. None off the SPM tasks, except the staff (Section 3.3.1) are applicable to this phase of the system metaphor.

4.2.3.2.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's system metaphor, with respect to SPM, in the monitoring phase and offer improvements to the process. The SPM shortcomings in this section (Staff) are not addressed by the system metaphor and should be applied to the process to improve it.

In the monitoring phase of the system metaphor the addition of the SPM of monitoring the staff (Section 3.3.1) is applicable to the system metaphor practice.

Section 3.3.1 should be applied to the practice during the first iteration to address SPM tasks for this practice.

The system metaphor is an important phase for helping guide the development team. More importantly, the system metaphor is the first phase of the project where the programmer, management, and clients work towards a common goal. The project manager needs to closely monitor the interaction between the staff and the clients. This process should the project manager asking:

1. Are the staff and clients engaged in the brainstorming session?
2. Do all the parties ask for clarification to questions and answers?
3. Are the staff member focusing on meaning or intent of the comments, rather that the delivery.
4. Is there a clear understanding and agreement on the priority of system functions, their development sequence and their value to the client?

4.2.3.3 – System Metaphor—Control & Adjust

4.2.3.3.1 – Software Project Management Addressed—The XP practice of creating the system metaphor covers none of the SPM techniques during it control and adjusting phase. The system metaphor practice is very simple in its concept and implementation. None off the SPM tasks, except the staff (Section 3.4.1) are applicable to this phase of the system metaphor.

4.2.3.3.2 – Software Project Management Shortcomings & Improvements—The controlling and adjusting phase of the system metaphor could use section 3.4.1 to direct and guide the staff. The system metaphor creation is a very simple task, and section 3.4.1 should be applied if the team is deadlocked and not moving forward. In summary, the system metaphor SPM shortcomings for the controlling and adjusting phase consist of addressing staffing needs (Section 3.4.1). All of the other SPM techniques are either not applicable to the system metaphor or handled by other XP practices.

4.2.3.4 – System Metaphor—Completion

4.2.3.4.1 – Software Project Management Addressed—The XP practice of creating the system metaphor includes none of the SPM techniques during its completion phase. Of the uncovered SPM techniques for the completion phase (Section 3.5) none of them are applicable to this XP practice. Any SPM completion tasks that arise for this process are handled by the planning game structure (Section 4.2.1).

4.2.3.4.2 – Software Project Management Shortcomings & Improvements—None of the SPM techniques defined in section 3.5 should be applied to the completion phase of the system metaphor process.

4.2.4 – Simple Design

This section will examine how the XP practice of having a simple design includes standard software project management tasks and techniques. The next four sections will examine how XP handles this practice throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases of software project management. Further subsections will identify how the simple design practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.4.1 – Simple Design—Estimate & Plan

4.2.4.1.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Software Development Process and Schedule) by the XP practice of simple design during the estimate and planning phase. The author will also identify those SPM tasks that are not applicable (Staff, Resources, Effort, and Deployment) to simple design during this phase.

The SPM task of estimating and planning for the software development process (Section 3.2.2) is addressed by the simple design practice (Section 2.8). Picking a design methodology and laying out the rules and steps to execute accomplish this. The next SPM task addressed by the simple design in this phase is the schedule (Section 3.2.6). However, this SPM task is handled in a minimalist way. The planning game's iteration and release planning process with spike solutions allow the team to work together to create a simple design using UML and CRC. This provides a place in the schedule but does not provide any insight into the length of time the design process will take. The planning and estimation of the schedule length for the simple design process is handled by grouping all the tasks in an iteration and using the iterations velocity to estimate, plan, monitor, adjust, and complete the schedule.

Estimating and planning for the staff (Section 3.2.1) for the simple design process is not applicable. The simple design process uses the labor cost per week of available staff resources to estimate and plan for what can be accomplished. The simple design process is addressed within the planning game practice. Any resource planning (Section 3.2.4) is handled by the planning game practice and is not applicable to the simple design practice. The simple design process does not require or apply effort estimating and planning practices (Section 3.2.5). The usage rate of resources associated with the simple design process is how the simple design process accounts for this SPM task. Estimating and planning for the deployment (Section 3.2.8) is not applicable to this phase of the simple design practice. Closely paired to the simple design practice is the planning game, where additions to the practice (Section 4.2.1) have accounted for the deployment estimating and planning for the simple design practice.

4.2.4.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's simple design, with respect to SPM, in the estimate and plan phase and offer improvements to the process. The SPM shortcomings in this section (Risk Assessment and Measurements) are not addressed by simple design and should be applied to the process to improve it.

The simple design practice does not account for any risk assessment (Section 3.2.3). The risk assessment process (Section 3.2.3) should be added to the simple design practice. Even the simplest of designs has risks that should be accounted for. The estimation and planning for risk in this phase should be modified to better fit the overall XP process. The software project manager can do this by including tasks that enable the simple design process to take place while identifying, estimating, and planning for risks.

In estimating and planning for risk in the simple design practice, risk number one is that no design is created. This can happen because the simple design process is only recommended when "…you aren't clear on just what to do, or if you would like a little reassurance, or if you would like a few additional ideas" (Jeffries, Anderson, and Hendrickson 2001, 70). Many programmers like to jump into the problem code and fix their way out. First, the simple design process needs to be defined and encouraged. The estimating and planning for risks associated with the design can be addressed. Below is a diagram depicting where in the XP process the simple design practice takes place (Figure 17).

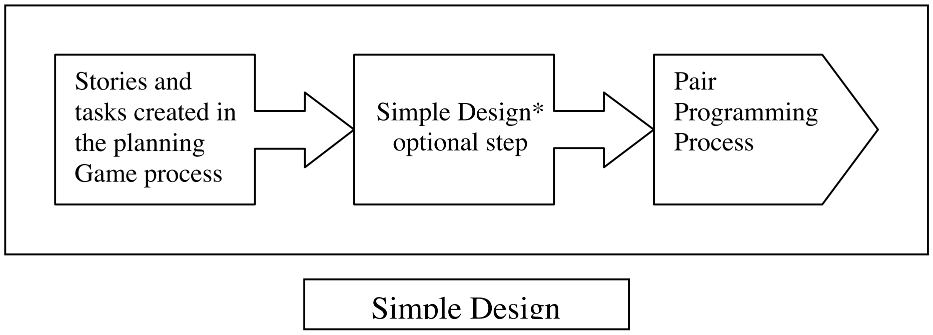

Stories and tasks created in the planning Game process
Simple Design* optional step
Pair Programming Process
Simple Design

Figure 17 – The simple design practice within the XP process.

Notice that the simple design process is optional and occurs just before the pair programming practice. The XP manager needs to require the simple design practice so that risk assessment can be done. When the planning game is done and the XP team members are starting the coding process, the common area where the team members gather to pair up is where the simple design process should start.

When starting the simple design process the XP team has to agree on a design method. Part of the coding standards planning and definition should include a standard design structure (CRC, UML, etc.). The simple design process, no matter how it is implemented, should fit on a 5 x 8 blank story card. Once the team has standardized how they will implement the simple design, these story cards should be placed next to each workstation where the pair programming process will take place. One side of the card is reserved for the simple design while the other side is reserved for any risks associated with the chosen design. Keep these design risks simple and to the point. Because the design process is optional in XP, the project manager should encourage the process, while not over loading the programming staff with more documentation. The practice of having the programmer write down some risk notes, in simple verbiage, on the back of each design card fits appropriately into the XP process.

The simple design practice does not have any way to estimate and plan for measurements. The SPM task of estimating and planning for measurements (Section 3.2.7) can and should be applied to the simple design practice. This is an overall effort to improve the accuracy of the schedule for its smaller components. Specifically measurements should be created to track the amount of effort in terms of design, if required for each story. This will add more accuracy and detail to the iteration planning and velocity planning process.

<u>4.2.4.2 – Simple Design—Monitor</u>

4.2.4.2.1 – Software Project Management Addressed—The simple design practice covers none of the SPM techniques during its monitoring phase. The system simple design practice is very simple in it concept and implementation. None off the SPM tasks, except monitoring measurements (Section 3.3.7), are applicable to this phase of the simple design practice.

4.2.4.2.2 – Software Project Management Shortcomings & Improvements—In the monitoring phase of the simple design practice the addition of the SPM task of monitoring the measurements (Section 3.3.7) is applicable to the simple design practice. Section 3.3.7 should be applied to follow up and track any measurements associated with the simple design practice. By monitoring the simple design practice the software project manager can increase or decrease emphasis on or off this optional task.

4.2.4.3 – Simple Design—Control & Adjust

4.2.4.3.1 – Software Project Management Addressed—The simple design practice includes none of the SPM techniques during its controlling and adjusting phase. The simple design practice is very simple in its concept and implementation. None off the SPM tasks except controlling and adjusting of the gathered measurements (Section 3.4.7) are applicable to this phase of the simple design practice. The coach (Section 2.19.2) in their role as lead architect is responsible for oversight and input into the simple design process. They control and adjust the process when needed. This is mainly done by reviewing the metrics and overhearing and interacting with the XP programming staff.

The controlling and adjusting SPM task is handled by the planning game and should not be applied directly to simple design because it is an optional practice. While the controlling and adjusting of the simple design's measurements are applicable to the process they are not covered by any SPM task in the XP process.

4.2.4.3.2 – Software Project Management Shortcomings & Improvements—The simple design practice is missing a mechanism for handling the SPM task of controlling and adjusting of the gathered measurements. Section 3.4.7 should be applied to the measurement process associated with the simple design practice. This SPM task should be applied in a way that the XP manager and coach can evaluate the effectiveness of the simple design process and determine if the team is following the simple design methodology.

As the lead architect the coach (Section 2.19.2) has the responsibility to oversee and adjust the design process. They do not need to be directly involved in each design session, but they should review the results and consult with the team when red flags for problems arise. Reviewed designs should be marked and recorded. If problems arise the coach should attach are refactoring note to the story/design card. The refactoring note should include the current problems with the simple design, desired improvement and goals or reasoning behind the updated design. Then the story card should be give back to the team. This process should be executed in a way that allows the team to leverage and learn from the coach's experience and knowledge.

4.2.4.4 – Simple Design—Completion

4.2.4.4.1 – Software Project Management Addressed—The simple design practice (Section 2.7) includes none of the SPM techniques during its completion phase. Of the missing SPM techniques for the completion phase (Section 3.5) none are applicable to this XP practice. Any SPM completion tasks that arise for this process are handled by the planning game structure (Section 4.2.1).

4.2.4.4.2 – Software Project Management Shortcomings & Improvements—None of the SPM techniques defined in section 3.5 should be applied to this phase of the simple design practice and no additional improvements are necessary.

4.2.5 – Testing

This section will examine how XP handles the testing practice with respect to standard software project management tasks and techniques. The next four sections will examine how XP handles the testing practice throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases of software project management. Further subsections will identify how the testing practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.5.1 – Testing—Estimate & Plan

4.2.5.1.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Staff – Roles, Schedule – Unit test, Measurements, and Deployment) by the XP practice of testing during the estimate and planning phase. The author will also identify those SPM tasks that are not applicable (Staff, Software Development Process, and Effort) to testing during this phase.

The XP testing practice addresses some (Sections 3.2.1.2 and 3.2.1.3) SPM tasks with regards to the staff. The testing practice is the only XP process that has a dedicated staff member (Section 2.18.3) who is not part of the programming staff. The XP tester is allocated at the beginning of the project and is responsible for supporting the programming staff with unit tests and, most importantly, working closely with the client to create the acceptance tests. Estimating the staff size (Section 3.2.1.1) is not applicable to the testing process. The staff is considered a fixed resource in XP. The XP process focuses on the cost per week for the staffing resource.

The next SPM task addressed in the estimating and planning phase is the schedule. The testing practice addresses the schedule through the planning game process (Section 4.2.1). The planning game process only addresses the unit test portion of the SPM tasks associated with the testing practice. The next section will address the acceptance testing for testing practice. The software development process (Section 3.2.2) estimation is not applicable to the testing practice. The testing practice is well defined and planning of the software development process is handled by other XP practices. Similar to estimating and planning staff size to match effort, estimating the testing effort (Section 3.2.5) is not applicable to the testing practice. The planning game process encompasses the effort of estimation and planning of the testing practice (Section 4.2.1).

The XP process addresses the planning and estimating of measurements within the testing practice. The coach addresses unit testing measurements (Section 2.18.5) by encouraging the team to follow the testing procedures. The unit tests (Section 2.9.1) are defined by a test-first process through pair programming (Section 2.11), which tries to ensure the process is followed. Acceptance tests are covered by defined metrics (Section 2.19.1) for the XP process.

The last SPM task, for this phase is the estimation and planning of deployment (Section 3.2.8). Deployment estimation and planning is handled through the planning game's stories. The deployment stories are an addition to the XP process. The testing practice verifies those deployment stories (Section 4.2.1.1.3) by running the deployment tests.

4.2.4.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's testing practice, with respect to SPM, in the estimate and plan phase and offer improvements to the process. The SPM shortcomings in this section (Risk Assessment, Resources, Schedule – Acceptance Tests) are not addressed by testing and should be applied to the process to improve it.

The XP process does address the need for risk assessment in the testing practice, although not completely. Story cards within the simple design, pair programming, and the planning game practices handle testing risks and assessment of those risks. Even with these systems in place to address risks the testing practice is lacking in three areas: testing tools, test execution (this is risk to the ability to run valid tests on the deployment environment), and staff risks. What tools the XP team uses to implement the testing practice and who is going to work with the client are critical to the success of the testing practice. Applying SPM risk assessment practices (Section 3.2.3) to these two areas will improve the estimating and planning phase of the testing practice. In addition to identifying the risks associated with the testing tools, the tools themselves need to be estimated and planned. Standard software project management resource estimation (Section 3.2.4) should be applied to the selection of the proper testing tools.

The scheduling for the acceptance testing in the testing practice is not defined or covered by the XP practice. Acceptance testing is the only XP practice that is not scheduled through the planning game process. The only guideline given for estimation and planning of the testing, effort, and schedule is that the programmer and the client should, "…get them [acceptance tests] to the programmers by the middle of each iteration" (Jeffries, Anderson, and Hendrickson 2001, 32). The tester and the client have the stories, from the iteration planning process (Section 2.5.2), which require acceptance tests, but no clear schedule. Standard project management scheduling practices (Section 3.2.6) can be applied to the estimation and planning of acceptance testing to improve it.

<u>4.2.5.2 – Testing—Monitor</u>

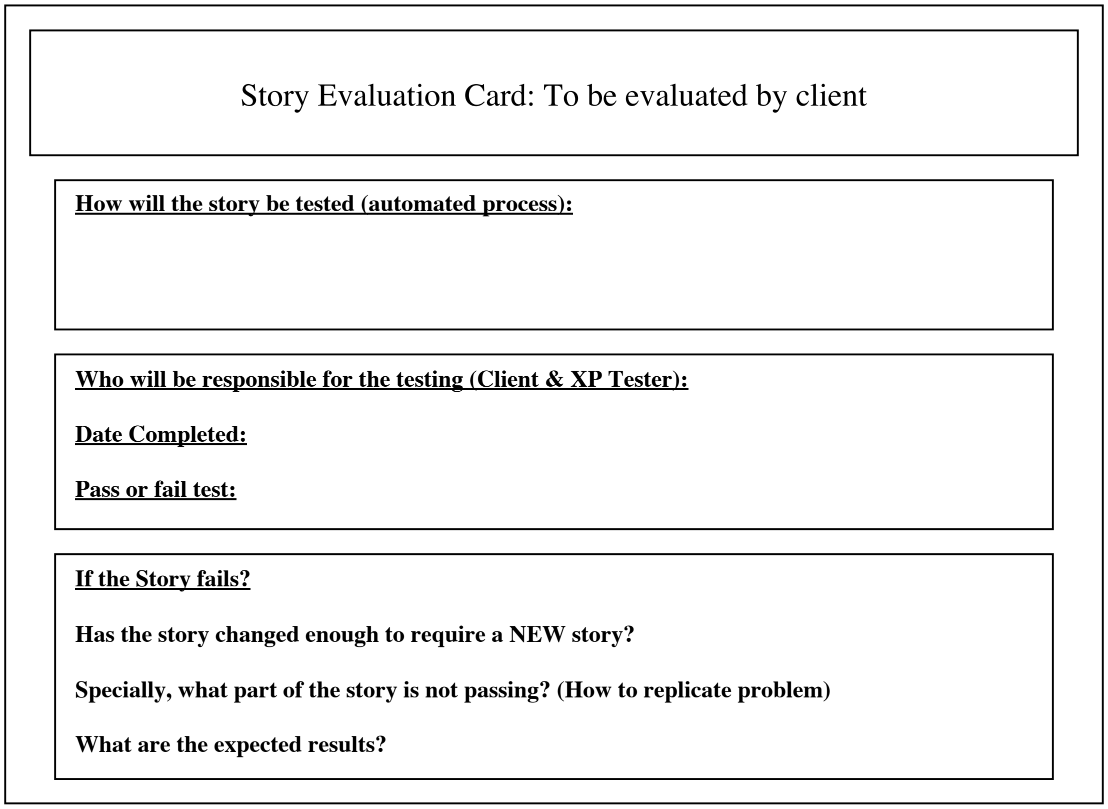
4.2.5.2.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Staff, Resources, Schedule – Unit tests, Measurements, and Deployment) by the XP practice of testing during the monitoring phase. There are no SPM tasks that are not applicable to testing during this phase.

The monitoring of the staff in the testing practice is addressed by the XP tasks associated with this practice. Within the testing practice, the tracker tracks the progress of the testing (Section 2.18.4) and the coach (Section 2.18.5) monitors the testing process. The coach is also responsible for monitoring the staff and activities, which includes the testing resources. The schedule for the unit tests is addressed by the planning game practice (Section 2.5). The XP team's tracker (Section 2.18.4) monitors measurements associated with the testing practice. The monitoring of the testing with regards to deployment is addressed by the planning game practice (Section 2.5) and the small release practice (Section 2.6).

4.2.5.2.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's testing practice, with respect to SPM, in the monitoring phase and offer improvements to the process. The SPM shortcomings in this section (Software Development Process, Risk Assessment, and Effort) are not addressed by testing and should be applied to the process to improve it.

The section addressing the estimating and planning of testing (Section 4.2.5.1) showed the need to add SPM tasks to address three risk assessment areas. The tools selected and the staff member assigned to the testing process need to be monitored with regards to risk; this is not covered by the testing practice and section 3.3.3 should be applied to these two areas. Risk assessment is not the only way to monitor tools. This phase does not address the SPM task of monitoring tools, hardware, and software. The SPM task in section 3.3.2 should be applied to fulfill this SPM need. The XP coach and manager should complete the SPM monitoring tasks associated with the software development process. Effort associated with the acceptance testing area should be monitored because it can become a gating issue for an iteration. The acceptance tests can follow a more traditional software project management process and can be addressed by applying section 3.3.5 to monitor the acceptance testing effort.

Acceptance testing seems to be the weak link in XP testing practice. The SPM tasks added to monitor the effort should be applied within the XP methodology. The effort required for the acceptance-testing task should also be monitored and documented for each story. This is because acceptance test are tightly coupled to each story. To track and monitor the status and effort of each acceptance test a story evaluation card should be attached to each story (figure 18). These cards should be attached when the programmer accepts each story during the iteration planning process. They should be updated and tracked by the tester and the client. The output of these story evaluation cards will be recycled into the iteration planning process in the planning game if new stories or story failures arise from the process.



Story Evaluation Card: To be evaluated by client

How will the story be tested (automated process):

Who will be responsible for the testing (Client & XP Tester):

Date Completed:

Pass or fail test:

If the Story fails?

Has the story changed enough to require a NEW story?

Specially, what part of the story is not passing? (How to replicate problem)

What are the expected results?

Figure 18 – An outline of story evaluation planning.

4.2.5.3 – Testing—Control & Adjust

4.2.5.3.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Staff, Resources, Schedule – Unit tests, Measurements, and Deployment) by the XP practice of testing during the control and adjust phase. The author will also identify those SPM tasks that are not applicable (Software Development, and Effort) to testing during this phase.

The testing practice addresses the controlling and adjusting of the staff (Section 3.4.1) through a combination of iteration planning, the coach, the tracker, and the manager (Sections 2.5.4, 2.18.5, 2.18.4, and 2.18.7). These XP practices address all of the unit test SPM tasks. The tracker and the tester handle controlling and adjusting of the staff with regards to the acceptance tests (Sections 2.18.3 and 2.18.4). The XP coach, the manager, and the tester address testing resources (Sections 2.18.5, 2.18.7, and 2.18.3). The client's (Sections 2.18.2 and 2.15) proximity to the testing staff and close working relationship allows them to help address any testing needs as they arise. Unit test scheduling needs are addressed through the planning game process (Section 2.5). Controlling and adjusting of the testing measurement is the job of the tracker (Section 2.18.4). The last SPM task that is addressed by testing practice in this phase is testing with regards to deployment. Additional stories are developed during the planning game to handle unit tests and acceptance tests with regards to deployment (Section 2.5.1).

In this phase of testing controlling and adjusting of the software development process (Section 3.4.2) is not applicable. Lastly, controlling and adjusting of the effort (Section 3.4.5) associated with the testing practice is not applicable. The testing practice focuses on the usage rate of resources over time and does not adjust the staff or resources based on effort. This allows changing only the functionality of the software during the planning game process.

4.2.5.3.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's testing practice, with respect to SPM, in the control and adjust phase and offer improvements to the process. The SPM shortcomings in this section (Risk Assessment) are not addressed by testing and should be applied to the process to improve it.

The XP testing practice puts in place the personnel to address many SPM tasks that could arise, but fails to say when and how. For the testing practice, in the controlling and adjusting phase, earlier sections (4.2.5.1 and 4.2.5.2) showed a need to estimate, plan, and monitor the testing execution, tools, and the team's tester. The XP team's manager (Section 2.18.7) and/or the team's coach (Section 2.18.5) should address the need to control and adjust risks associated with the testing practice. The SPM tasks with regards to risk assessment (Section 3.4.3) should be applied to this phase of the testing practice. The testing practice has enough structure to adjust to any testing needs, but the coach, the manager, and the programmers should monitor the acceptance test progress. The acceptance test execution is the weak link in the XP testing practice.

4.2.5.4 – Testing—Completion

4.2.5.1.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Deployment) by the XP practice of testing during the completion phase. The author will also identify those SPM tasks that are not applicable (Staff, Software Development Process, Schedule, and Effort) to the testing phase.

The only SPM tasks that are addressed by the testing practice in this phase is the deployment completion (Section 3.5.8). This phase of the testing process is completed when the entire acceptance test list runs correctly and the product is complete. The test results may not get accepted until the final iteration, but the acceptance tests are the system and client's signoff for the product.

The completion of the testing practice with regards the staffing (Section 3.5.1) SPM tasks is not applicable to this practice. Staffing SPM tasks for this phase are covered by the planning game practice (Section 4.2.1). The testing practice software development (Section 3.5.2) Software project management tasks are not applicable to this phase. XP generally does not advocate documenting the effectiveness of any of the12 practices overall. Instead it tries to react in real-time to development problems that arise.

If any documentation about the software development practice for the testing practice is recorded, it would be covered by the planning game process (Section 4.2.1). The administrative closeout for the schedule (Section 3.5.5) is not applicable to the testing practice. Overall schedule and SPM tasks associated with the schedule are handled by the planning game (Section 4.2.1). Tracking and recording of the testing effort (Section 3.5.6) is not applicable to the testing practice. Effort in the XP practice is estimated, monitored, and adjusted, by the planning game process. The planning game addresses SPM tasks associated with completion effort (Section 3.5.6).

4.2.4.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's testing practice, with respect to SPM, in the completion phase and offer improvements to the process. The SPM shortcomings in this section (Risk Assessment, Resources, and Measurements) are not addressed by testing and should be applied to the process to improve it.

Testing risks (Section 3.5.3) need to be documented and cycled back into the planning game process to improve scheduling and help prediction of overall costs. The SPM tasks associated with risk completion are not handled by the testing practice. These tasks should be added and the result should be sent into the planning game's completion phase (Section 4.2.1.4). Risks should include the tools and personnel used in the testing process.

The freeing up of testing resources and upgrading of tools are two SPM tasks (Section 3.5.4) that are not addressed by the testing practice. With the additional SPM tasks of estimating, monitoring, and controlling of the critical testing tools and tester position, the application of section 3.4.5 to the completion phase of testing could improve the testing process in future projects or iterations. The testing practice doesn't account for the necessity to aggregate and compile the gathered measurements (Section 3.5.7). SPM tasks (Section 3.5.7) associated with completion of the measurement are not address by this phase. Due to the critical nature of unit test and acceptance tests applying Section 3.5.7 is suggested. In applying Section 3.5.7 to this phase of testing the XP team's future projects will have better effort estimation, tools, and personnel information for the planning game process.

The most important addition in this section is the addition of freeing up the client resource. Providing an employee as an onsite client throughout an XP process can be difficult for the client's organization. Because the client plays a critical role in the testing practice, along with the other XP practices, the team and management should strive to make their time with the client as painless as possible. The project manager should take the time necessary to ensure the on-site client can resume their primary position as soon as possible.

4.2.6 – Refactoring

This section will examine how the XP practice of refactoring addresses standard software project management tasks and techniques. The next four sections will examine how XP address the refactoring practice throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases of software project management. Further subsections will identify how the refactoring practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

The refactoring practice should be considered a sub-practice of the pair programming practice and planning game practice.. All of the XP practices are interconnected but the refactoring does not significantly influence any of the other practices but it adds definition and structure to the actual coding process. As the next four sections will show, most of the standard project management practices are covered by other XP practices or do not apply to the refactoring process.

4.2.6.1 – Refactoring—Estimate & Plan

4.2.6.1.1 – Software Project Management Addressed—This section will identify that none of the standard SPM tasks are addressed by the refactoring practice during the estimate and planning phase. However, none of these SPM tasks are applicable to the refactoring practice during this phase.

Staff is considered a fixed resource in the XP process. The refactoring practice is used to direct the current staff, not to estimate and plan for them. Adjustments to the software development process with regards to refactoring are handled by the planning game and the pair programming practices. Risk planning for the refactoring practice should be covered when the management examined the XP software development practice. Refactoring is a coding and design activity. The planning game and pair programming practices handle any estimates and planning needed by the refactoring process. Defining and integrating refactoring into the schedule is handled by the planning game when the programmers estimate story tasks. Planning and estimating effort is not applicable because XP focuses on the rate of resource usage over time instead of adjusting resource to match effort. The refactoring practice does not affect or influence the deployment process and thus the SPM task associated with it are not applicable to the refactoring practice.

4.2.6.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's refactoring, with respect to SPM, in the estimate and plan phase and offer improvements to the process. The SPM shortcomings in this section (Measurements) are not addressed by refactoring and should be applied to the process to improve it.

The refactoring process of very small incremental improvements is highly suited to software metric application. The standard XP process does not track or monitor the refactoring process. The coach and the pair programming process support its execution. The SPM task of estimating and planning for measurements (Section 3.2.7) is applicable and should be applied to the refactoring process. The refactoring practice by definition applies unit tests to each incremental change. Specifically the new unit tests written just for refactoring and separate unit tests are written for defects. Knowing how often, how many, and the motivation behind each refactoring session will give the coach and manager the information needed to evaluate the effectiveness of the refactoring process.

Planning for and tracking of measurements associated with the refactoring process should be executed in a way that adds minimal effort for the XP staff. The refactoring process is started and tracked with unit tests. Any measurement planned should help the coach and the manager understand when and why the "code smells" (Wake 2002, 27). Unit tests are implemented as classes that can be inherited and extended to meet current project needs. Using inheritance, the XP team's unit tests should be extended to capture who is refactoring the code and for what reason. Two simple enumeration classes that are set in each unit test class can accomplish this task. The first enumeration should list every combination of programmers in pairs. This should only be done with a small team, for a 10 person programming staff this list would consist of 45 different combinations. The next enumeration should list driving factors for the refactoring. Start with a simple list and have the programming team add factors where needed. Examples could include:

- Method too large
- Magic numbers
- Classes too large, need to be decomposed
- Coding standard not followed
- Incomplete or inaccurate unit test
- Improvements in speed
- Additional error checking

These two additions to the unit test classes will enable the XP team to measure who is refactoring the code and why the refactoring effort was done. This could lead to better coding practices that reduce the need for refactoring, thus reducing cost and schedule length.

4.2.6.2 – Refactoring—Monitor

4.2.6.2.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Measurements) by the XP practice of refactoring during the monitoring phase. The author will also identify those SPM tasks that are not applicable (Staff, Software Development, Risk Assessment, Resources, Effort, Schedule, and Deployment) to refactoring during this phase.

The SPM tasks for staff, software development process, risk assessment, resources, effort, schedule and deployment, are not applicable to the refactoring practice. As stated in the estimating and planning phase, all of the SPM tasks except measurements are handled by other XP practices. These include the planning game, pair programming and testing practices.

As shown in other sections, the XP process addresses the monitoring phase of SPM tasks more completely than the estimating and planning phase. The refactoring practice is an exception. The monitoring of the gathered measurements (Section 3.3.7) is the only SPM task that is applicable to this practice. In addition, the XP process addresses the monitoring of the refactoring practice measurements. The XP team's tracker (Section 2.18.4) and the coach (Section 2.18.5) gather and monitor measurements during refactoring.

4.2.6.2.2 – Software Project Management Shortcomings & Improvements—In this phase of the refactoring practice either the SPM tasks are not applicable or are addressed by the XP and refactoring process. This leaves no software project management shortcoming for the monitoring phase of the refactoring practice.

4.2.6.3 – Refactoring—Control & Adjust

4.2.6.3.1 – Software Project Management Addressed—This section shows that the refactoring practice covers none of the SPM techniques directly during its controlling and adjusting phase. Several of the SPM techniques are covered or addressed (Staff, Schedule) by other XP practices that encompass the refactoring practice. The author will examine those SPM tasks that are not applicable (Software Development Process, Risk Assessment, Resources, Effort, Schedule, and Deployment) to refactoring during this phase.

As stated, none of the SPM techniques are directly coved by the refactoring practice. The refactoring practice is highly coupled with the pair programming practice and the planning game practice and falls under the guidance and watchful eyes of the coach and the tracker. Two SPM tasks, controlling and adjusting the staff (Section 3,4,1) and controlling and adjusting the schedule (Section 3.4.6), are handled by other XP practices. The staff's SPM tasks are covered by the planning game practice (Section 4.2.1), the coach (Section 2.18.5), and self adjusted by the programmers themselves (Section 2.18.1). Almost all scheduling in any XP practices is handled by the planning game practice (Section 4.2.1). Certainly refactoring takes time on the project schedule, but refactoring is impromptu updates or fixes to the code and the XP practice accounts for these additions to the schedule by the project's velocity and the story task estimation process in the planning game (Section 4.2.1). Additional measurements tracking or monitoring the refactoring process or velocity should flow back to the planning game practice to better estimate, monitor, adjust, and finish the project's schedule.

Adjusting the software development (Section 3.4.2) process in not applicable to the refactoring practice. Risk assessment (Section 4.4.3) is not applicable to the refactoring practice. Risk is handled by the testing (Section 2.9), planning game (Section 2.5) and pair programming (Section 2.11) practices. Resource adjustment (Section 3.4.4) is not applicable during the refactoring practice because the needed resources are adjusted and controlled by the planning game, testing, XP manager, and XP coach. Effort (Section 3.4.5) associated with the refactoring practice is handled by the planning game process and is not applicable to the refactoring practice. Lastly, deployment (Section 3.4.8) with regards to refactoring is not applicable to the practice.

4.2.6.3.2 – Software Project Management Shortcomings & Improvements— Controlling and adjusting of the refactoring process only has one SPM task that is not addressed by the practice; the adjustment of measurements (Section 3.4.7). While the overall XP process has people in place to control and adjust measurements (Sections 2.18.5 and 2.18.4), these processes are not specifically applied to the controlling and adjusting of refactoring measurements. With the addition of measurements specific to the refactoring practice (Section 4.2.6.1) additional SPM tasks associated with controlling and adjusting measurements (Section 3.4.7) should be applied to the refactoring practice.

4.2.6.4 – Refactoring—Completion

The XP practice of refactoring (Section 2.10) covers none of the SPM techniques during its completion phase. Of the uncovered SPM techniques for the completion phase (Section 3.5), none of them are applicable to this XP practice. The testing and pair programming practices handle any SPM completion tasks that arise for this (Sections 2.9 and 2.11).

4.2.6.4.1 – Software Project Management Addressed—None of the SPM techniques defined in section 3.5 should be applied to this phase of the refactoring process. The SPM tasks associated with the staff (staffing down, allocating staff to the assessment process, and allocating staff for support) are addressed by the planning game practice (Section 4.2.1) and pair programming (Section 2.11) practice and are not applicable to the refactoring practice. Evaluation of the completion of the refactoring software development practice (Section 3.5.2) is not applicable to this practice. If management wishes to evaluate this practice during the software development process, the completion phase of the planning game is the appropriate place. Resources (Section 3.5.4) associated with refactoring do not need to be released, because higher processes like the planning game and pair programming are using them. All scheduling (Section 3.5.5) associated with completion of the schedule is handled by the planning game and is not appropriate to the refactoring practice. Effort (Section 3.5.6) is handled by the planning game (Section 4.2.1.4) and is not appropriate for the refactoring practice. Knowing when stop refactoring is a measure of the story task the programmers are working on. The coach and the tracker watch and support the pairs in the refactoring completion. Refactoring is completed when the code has no duplicate code and only has the necessary classes and methods. Measurements (Section 3.5.7) are aggregated and tallied by the team's tracker (Section 2.18.4). The planning game process handles any achieving of measurements (Section 4.2.1.4). Lastly the refactoring practice is not associated with any deployment completion task and SPM completion tasks (Section 3.5.8) are not appropriate for the refactoring practice.

4.2.6.4.2 – Software Project Management Shortcomings & Improvements—The completion phase of the refactoring practice has no software project management shortcomings. All of the SPM tasks associated with the completion phase are either not applicable or covered by other XP practices.

4.2.7 – Pair Programming

This section will examine how XP handles the pair programming practice with respect to standard software project management tasks and techniques. The next four sections will examine how XP handles the pair programming practice throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases of software project management. Further subsections will identify how the pair programming practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.7.1 – Pair Programming—Estimate & Plan

4.2.7.1.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Staff – Roles) by the XP practice of pair programming during the estimate and planning phase. The author will also identify those SPM tasks that are not applicable (Size, Software Development Process, Effort, and Deployment) to pair programming during this phase.

The pair programming process lays out rules for execution of the practice, but offers very little in the estimating and planning phase. The only SPM task addressed by the pair programming practice is assigning staff roles (Section 3.2.1.3). The manager addresses this SPM task at the beginning of an XP project as defined by the XP practice (Section 2.18).

Some of the SPM tasks are not applicable to the pair programming practice. First, the pair programming practice does not estimate the staff size to match effort (Section 3.2.1.1). The pair programming practice focuses on the effort expenditure over time with respect to staff members for this practice. Specifying what type of software development process (Section 3.2.2) is used for the pair programming practice is not applicable to the XP practice. The XP process outlines the rules and steps to execute the pair programming practice (Section 3.2.2). The estimation and planning of project effort (Section 3.2.5) is not applicable to the pair programming practice. The pair programming practice is not responsible for effort estimation in the XP practice. Effort with regards to the pair programming practice is handled through the planning game process (Section 4.2.1). Lastly, estimation and planning of deployment tasks (Section 3.2.8) associated with the pair programming practice is not applicable to the practice.

4.2.7.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's pair programming, with respect to SPM, in the estimate and plan phase and offer improvements to the process. The SPM shortcomings in this section (Development, Risk Assessment, Resources, Schedule, and Measurements) are not addressed by pair programming and should be applied to the process to improve it.

One of the benefits of the pair programming practice is to balance or share knowledge between team members. The XP process expects this to happen without planning for it and does not address this SPM task in the pair programming practice. The XP manager and the coach need plan the SPM task (Section 3.2.1.4) with respect to the pair programming team's skills and development at the beginning of an XP project. Before the start of the planning game the manager and the coach need to compile a system for addressing the team's programming development. This staff development plan should be in accordance to SPM outline in Section 3.2.1.4. How section 3.2.1.4 is applied to pair programming will vary per project and the coach and the project manager have discretion on individual, pairs and team development tasks.

The pair programming practice does not address any risk (Section 3.2.3) associated with the pair programming practice. The coach (Section 2.18.5) and the XP manager (Section 2.18.5) should apply the SPM tasks associated with risk assessment (Section 3.2.3) towards the pair programming practice. Specifically section 3.2.3 should be applied towards team culture, programming knowledge, communication skills and domain knowledge. These four areas were chosen because many of the XP practices will be executed and carried out by the programming staff. How the team works together, and works with the clients will determine the success or failure of an XP project. The majority of risks in this phase are associated with communication: between the pairs, the clients, the stories and tools. In the estimating and planning phase of each release the coach and the project manager should take the time to plan for the risks associated with the pair programming practice.

In addition to the missing SPM task of risk assessment the pair programming practice is missing the SPM task of estimating and planning for resources (Section 3.2.3) associated with the pair programming practice. During the risk evaluation process the coach and the project manager should apply the estimation and planning techniques from section 3.2.3 to define and acquire the resources needed for the team to be successful.

Standard project management tasks associated with the schedule (Section 3.2.6) are missing in the pair programming practice. When the team pairs and when the development gets scheduled is handled by the planning game practice. The estimation and planning for the pair programming practice needs to support the additional SPM tasks added to the practice (staff development, risks). Any staff training sessions associated with the pair programming practices, staff development, or communication improvement need to be funneled into the planning game practice. Following the SPM techniques outlined in section 3.2.6 a team story card needs to be created with tasks that need to be scheduled. These stories will focus on improving the overall effectiveness of pair programming.

The pair programming practice (Section 2.11) does not address estimating and planning for any measurements (Section 3.2.7). The SPM task associated with measurement (Section 3.2.7) should be added as task for the tracker (Section 2.18.4) and the coach (Section 2.18.5). Measurement in this phase should focus on which pairs have deployed the product, link pairs and individuals programmer to refactoring, and associate stories to pairs and/or individuals. Deployment is an important process for any software development practice. In an XP project every team member needs to understand and be able to deploy the product. The pair programming practice is a good point to understand

and track the team's progress with regards to deployment. The pair programming practice (Section 2.11) lays out strict rules for how the pair programming works, but no mechanisms to verify or track it. This should be the focus for adding measurement planning (Section 3.2.7) in this phase.

All of the additional SPM techniques added to the pair programming practice have a common problem of when to integrate the well defined SPM tasks into the pair programming process. A suggested approach is to integrate these SPM tasks into the XP workflow with the team story card. The XP manager and the coach are the primary creators of the team story cards. The XP team's tracker (Section 2.18.4) can determine when a team story card is needed. These stories are placeholders for SPM tasks that in a traditional XP process would not be addressed. By creating SPM tasks associated with the pair programming practice and adding them to the planning game process it adds the ability to address SPM tasks with communication, courage, simplicity and quick feedback. The story card can use the normal story card format but be differentiated by some factor; specifying a different color is the most obvious. Using the same story format, the purpose of team story cards is to express the desired outcome. For example, "Tim and John want to pair together to work on saving xml data to a PostgreSQL database. But neither Tim nor John have not experience with PostgreSQL." This is an example of when a training session or workshop needs to be scheduled. Professional development and knowledge transfer takes time and effort. The introduction of the team story card is relevant within the programming estimating and planning section because of the importance of the pair programming practice. Team culture, experience, and communication are critical to the success of an XP team. The XP process asserts that one of its strengths is that it provides an enjoyable work process. The team story card integrates SPM tasks in an XP way.

4.2.7.2 – Pair Programming—Monitor

4.2.7.2.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Staff, Resources, Effort, Schedule, and Measurements) by the XP practice of pair programming during the monitoring phase.

Those SPM tasks that are not applicable (Staff – Size and Deployment) to pair programming during this phase will be identified.

Staff culture and roles (Section 3.3.1) are addressed by the pair programming process through the coach (Section 2.18.5) and self-monitored by the programming staff (Section 2.18.1). Monitoring the pair programming staff's professional development is covered by additions to the pair programming estimating and planning phase (Sections 4.2.7.1.2 & 4.2.7.1.3). By adding the team story into the planning game process, the planning game process monitors its progress during the monitoring phase. The only SPM task associated with the staff that is not applicable to the pair programming practice is matching staff size to effort (Section 3.3.1) because staff size is a fixed resource within the pair programming practice.

The planning game process monitors the effort associated (Section 3.3.5) with the pair programming practice (Section 2.5). Pair programming resources (Section 3.3.4), as they affect the staff assignment, is also addressed by the planning game practice (Section 2.5) and the pair programming practice (Section 2.11). The XP manager and the coach monitor the hardware, software, and environmental resources (Section 2.18.7). The pair programming practice monitors its schedule (Section 3.3.6) through the planning game process (Section 2.5). The last SPM task that is covered by the pair programming practice is monitoring of the gathered measurements (Section 3.3.7). This task is addressed and handled by the XP teams tracker (Section 2.18.4). Monitoring of the deployment (Section 3.3.8) is not applicable to the pair programming practice. The deployment tasks and their monitoring of are addressed by the planning game practice.

4.2.7.2.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's pair programming, with respect to SPM, in the monitoring phase and offer improvements to the process. The SPM shortcomings in this section (Software Development Process and Risk) are not addressed by pair programming and should be applied to the process to improve it.

The pair programming practice does not address SPM tasks associated with monitoring the software development process (Section 3.3.2). The coach in the XP practice does monitor the resources (tools, software, and hardware) but just to make sure the project is moving forward. Monitoring of the software development process looks for improvements in these areas. Thinking critically about how the team's resources affect the software development process (Section 3.3.2) should be added to the pair programming practice. The XP manager (Section 2.18.7) can address this SPM need without disrupting pair programming or the overall XP process.

The monitoring of risk (Section 3.3.3) associated with the pair programming practice is not addressed by the practice. In the estimating and planning section associated with the pair programming practice this thesis identified the need to plan for risks that are tightly coupled to the pair programming practice. These risks need to be monitored by the coach (Section 2.18.5) and the XP manager (Section 2.18.7).

4.2.7.3 – Pair Programming—Control & Adjust

4.2.7.3.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Staff, Resources, and Effort) by the XP practice of pair programming during the control and adjust phase. This section identifies those SPM tasks that are not applicable (Deployment and Schedule) to pair programming during this phase.

The pair programming practice is largely self-managed by the programmers (Section 2.18.1). The XP coach (Section 2.18.5) supports and guides the programmers to ensure the practice is followed correctly. The XP manager (Section 2.18.7) is the last level of pair programming management. The combination of these three levels of management cover the SPM task associated with controlling and adjusting the staff (Section 3.4.1). Adjusting of resources (Section 3.4.4) is the responsibility of the XP manager (Section 2.18.7); where the tracker (Section 2.18.4) and the coach (Section 2.18.5) bring the resources needs to them. The programmers handle the effort (Section 3.4.5) associated with controlling and adjusting for the pair programming practice (Section 2.18.1) and the practice (Section 2.11); asking for help and changing pairs is how the pair programming practices adjusts for varying effort.

Deployment (Section 3.4.8) and scheduling (Section 3.4.6) within the controlling and adjusting phase is not applicable to the pair programming practice. Deployment and scheduling should be handled through planning game process (Section 2.5).

4.2.7.3.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's pair programming, with respect to SPM, in the control and adjust phase and offer improvements to the process. The SPM shortcomings in this section (Software Development Process, Risk Assessment, and Measurements) are not addressed by pair programming and should be applied to the process to improve it.

The software development process in the controlling and adjusting phase (Section 3.4.2) examines the effectiveness of the pair programming rules and makes adjustments where needed. The pair programming practice allows for this to happen (Coach & Manager), but puts no process or responsibility in place to account for this SPM task. This process can and should be added to the pair programming practice. The pair programming practice does not address risk adjustment (Section 3.4.3). Risk associated with the XP team pair can have a major effect on the overall success of an XP project. The XP manager and the coach should apply Section 3.4.3 to control and adjust risks in this phase of pair programming. In the estimation and planning phase of the pair programming (Section 4.2.7.1) practice this paper showed a need to add measurements to the pair programming practice. These measurements may need to be adjusted per project and per team. The pair programming practice does not address this SPM task (Section 3.4.7) and it should be added to improve the process.

4.2.7.4 – Pair Programming—Completion

4.2.7.4.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of pair programming during the completion phase. Those SPM tasks that are not applicable (Software Development Process and Deployment) to pair programming during this phase will be identified.

The pair programming practice in the completion phase addresses none of the SPM tasks. Overall, XP practices rarely address any completion tasks. The SPM software development tasks (Section 3.5.1) are not applicable to the pair programming practice. Deployment completion tasks (Section 3.5.8) are not applicable to the pair programming practice. Neither of these SPM tasks fit in the pair programming tasks or timeline; they can be addressed in other XP practices where they would fit more organically into the process.

4.2.7.4.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's pair programming, with respect to SPM, in the completion phase and offer improvements to the process. The SPM shortcomings in this section (Staff, Risk, Resources, Effort, Schedule, and Measurements) are not addressed by pair programming and should be applied to the process to improve it.

Staffing (Section 3.5.1) SPM tasks are not addressed by the pair programming practice. This task is applicable to the practice and is an area needing improvement. However, the planning game should address this need. With improvements to the planning game practice (Section 4.2.1) this SPM need is covered. Completing and recording any risk associated with the pair programming practice (Section 3.5.3) is not addressed by the pair programming practice. These SPM tasks should be added the manager's tasks to improve the pair programming practice. This SPM task is added to record and avoid any future problems in new projects. The pair programming practice is essential to the XP process; to not learn from the perceived or materialized risks would only hurt the team's long-term success.

Resources (Section 3.5.4) for the pair programming practice are not addressed. Resource completion is handled by additions to the planning game practice (Section 4.2.1). No additional improvements are need here.

Measurements (Section 3.5.6) are also not addressed by the pair programming practice. These SPM tasks are covered by a combination of improvement to the XP process. The planning game (Section 4.2.1) estimates overall effort, and the testing practice (Section 4.2.5) captures the pair's testing effort. Lastly, the refactoring practice

(Section 4.2.6) captures the pair's refactoring effort. The pair programming practice does not directly address measurements (Section 3.5.7) associated with the pair programming completion. These additions to the XP process (Section 4.2.1) cover this SPM need.

4.2.8 – Collective Ownership

This section will examine how the XP practice of collective ownership addresses standard SPM tasks and techniques. The next four sections will examine how collective ownership addresses SPM tasks throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases. Further subsections will identify how the collective ownership practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.8.1 – Collective Ownership—Estimate & Plan

4.2.8.1.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of collective ownership during the estimate and planning phase. Those SPM tasks that are not applicable (Staff, Software Development Process, Risk Assessment, Resources, Effort, and Deployment) to collective ownership during this phase will be identified.

Collective ownership addresses none of the SPM tasks laid out in Chapter 3. From a SPM perspective, the collective ownership practice should be considered a component or a defined policy of the pair programming practice. All of the SPM tasks, except the schedule and measurements, are addressed by the collective ownerships parent practices (pair programming and the planning game).

4.2.8.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's collective ownership, with respect to SPM, in the estimate and plan phase and offer improvements to the process. The SPM shortcomings in this section (Schedule and Measurements) are not addressed by collective ownership and should be applied to the process to improve it.

The first SPM shortcoming associated with the estimating and planning phase of the collective ownership practice is the schedule (Section 3.2.6). The collective ownership practice can be considered a guideline for its parent component, the pair programming practice. Although, neither the pair programming practice nor the collective ownership practice defines when or where this guideline will be communicated to the team. The XP team needs to be informed and instructed on the software development practices and polices. This task needs to be scheduled. The scheduling of communication of XP policies could be handled following the SPM guideline in section 3.2.6 and implanted by using a team story card referenced in the pair programming section (4.2.7).

Collective ownership does not account for any measurements (Section 3.2.7). To be able to monitor this pair programming guideline, the tracker and the coach must use a metric to track the practice. Section 3.2.7 should be applied to cover this need. These measurements do not require a formal process, although the tools are in place to do so (Section 4.2.5). At this stage the coach and the tracker should specify some measurements to make sure that the natural separation of coding skill areas and interest do not inhibit the collective ownership practice. Separation or divisions in what code the individuals within the team work on can lead to team members blaming each other for failures. The collective ownership practice is the part of the XP process that cultivates a team atmosphere. The addition of SPM tasks (Section 3.2.7) should support this need.

The XP process defines the collective ownership practice, along with the other practices, but no clear division of labor is outlined for them. Applying SPM techniques to them will be ineffective if these tasks are not assigned to a role or roles in the process. The establishment of collective ownership through specific tasks should be a shared responsibility between the tracker (Section 2.18.4) and the coach (Section 2.18.5).

4.2.8.2 – Collective Ownership—Monitor

4.2.8.2.1 – Software Project Management Addressed—The XP practice of collective ownership (Section 2.12) covers none of the SPM techniques during its monitoring phase. The collective ownership practice is very simple in its concept and implementation. None off the SPM tasks, except monitoring staff (Section 3.3.1) and measurements (Section 3.3.7), are applicable to this phase of collective ownership. The coach should instill a staff culture that promotes and encourages the collective ownership. Measurement and staff monitoring should be used to support this process.

4.2.8.2.2 – Software Project Management Shortcomings & Improvements—The collective ownership practice defined by the XP process (Section 2.12) does not address the need to monitor its implementation. No additional SPM tasks need to be assigned to the collective ownership practice in this phase. The missing SPM tasks (Staff – section 3.3.1 and Measurements – Section 3.3.7) are covered by additions to the collective ownership parent components of pair programming (Section 4.2.7) and the planning game (Section 4.2.1).

4.2.8.3 – Collective Ownership—Control & Adjust

4.2.8.3.1 – Software Project Management Addressed—The XP practice of collective ownership (Section 2.12) covers none of the SPM techniques during its control and adjust phase. The collective ownership practice is very simple in its concept and implementation. As a supporting guideline for other XP practices (pair programming and the planning game) almost all the SPM tasks are covered by the parent practices. None of the SPM tasks, except the staff (Section 3.4.1) are applicable to this phase of the collective ownership practice.

4.2.8.3.2 – Software Project Management Shortcomings & Improvements—In the controlling and adjusting phase of the collective ownership practice, section 3.4.1 should be used to direct and guide the staff. Section 3.4.1 should be applied if the team is not dividing up the programming task reasonably and the manager is not isolating or assigning portions of the code to specific team members. This SPM task will be added to the coach's and the tracker's responsibilities. In summary, the collective ownership SPM shortcoming for the controlling and adjusting phase consists of addressing staffing needs (Section 3.4.1).

4.2.8.4 – Collective Ownership—Completion

4.2.8.4.1 – Software Project Management Addressed—The XP practice of collective ownership (Section 2.12) covers none of the SPM techniques during its completion phase. Of the uncovered SPM techniques for the completion phase (Section 3.5) none of them are applicable to this XP practice. The pair programming practice (Section 4.2.7) and the planning game practice (Section 4.2.1) handle all of the SPM completion tasks that arise for the collective ownership practice in the completion phase.

4.2.8.4.2 – Software Project Management Shortcomings & Improvements—None of the SPM techniques defined in section 3.5 should be applied to this phase of the collective ownership practice and no additional improvements are recommended.

4.2.9 – Continuous Integration

This section will examine how the XP practice, continuous integration, addresses standard software project management tasks and techniques. The next four sections will examine how continuous integration addresses SPM tasks throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases. Further subsections will identify how the continuous integration practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.9.1 – Continuous Integration—Estimate & Plan

4.2.9.1.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Risk) by the XP practice of continuous integration during the estimate and planning phase. Those SPM tasks that are not applicable (Staff and Software Development Process) to continuous integration during this phase will be identified.

Most of the guideline XP practices (40-hour workweek, system metaphor, coding standards, refactoring, collective ownership, continuous integration, small releases, and simple design) define the continuous integration practice, but little is written on how to plan, monitor, or adjust and finish. The only SPM task that is addressed by the continuous integration practice is risk assessment (Section 3.2.2). Slow merges and lost changes when releasing code are identified as risks and the XP practice offers solution and avoidance techniques (Section 2.13.2). The SPM techniques associated with the staff (Section 3.2.1) and the software development process (Section 3.2.2) are not applicable to this practice.

4.2.9.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's continuous integration, with respect to SPM, in the estimate and plan phase and offer improvements to the process. The SPM shortcomings in this section (Resources, Effort, Schedule, Measurements, and Deployment) are not addressed by continuous integration and should be applied to the process to improve it.

Continuous integration is highly coupled with building and deploying the project's deliverables. This can require build scripts, like MAKE or ANT, and possibly additional software. Estimating and planning for these resources (Section 3.2.4) is not addressed in the continuous integration practice. This practice should be added to prepare the team and XP project for the continuous integration practice. The estimating and planning of effort (Section 3.2.5) is mostly not applicable to the XP practice, but due to the complexity of the continuous integration practice it is necessary. Effort estimation for this practice should be applied to the setup of the base build and deployment solution.

Tightly coupled with effort is the planning of the schedule (Section 3.2.6). Section 3.2.6 should be applied to encompass planning of the schedule to take into account the continuous integration practice. Measurements (Section 3.2.7) are not addressed by this practice. The continuous integration practice has the team adding, building and deploying their code many times a day. The time associated with this XP practice could limit the amount of time the pairs have to work on stories. The estimation and planning of measurements (Section 3.2.7) should focus on creating measurements to monitor the overall hours associated with the continuous integration practice.

Deployment estimation (Section 3.2.7) is not directly addressed by the continuous integration practice. Section 3.2.7 should not be added to improve this practice; the planning game (Section 4.2.1) has added this SPM task to encompass the continuous integration need.

How these SPM techniques are integrated into the continuous integration practice is up to the project manager. The team story card (introduced in section 4.2.7) can be used to blend this SPM need into the structure of the XP process. The product of the team story card should be a specify procedure for the continuous integration. If changes need to be made in the process a new team story card can be addressed in the next iteration, or release planning session.

<u>4.2.9.2 – Continuous Integration—Monitor</u>

4.2.9.2.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of continuous integration during the monitoring phase. Those SPM tasks that are not applicable (Staff, Risk Assessment, and Measurements) to continuous integration during this phase will be identified.

The continuous integration practice addresses the SPM tasks associated with monitoring the staff (Section 3.3.1). The first level of staff monitoring comes from the pairs themselves (Section 2.11). Next, the tracker (Section 2.18.4) monitors the programming process, which includes the continuous integration practice. Lastly the XP coach (Section 2.18.5) monitors the continuous integration practice to verify that the guideline is being followed. While the XP process does not specifically address the monitoring of risk for the continuous integration practice, it does put the people and processes in place to accomplish this. Risk assessment (Section 3.3.3) and measurements (Section 3.3.7) are covered by the same system that monitors the staff, the pairs, the tracker, and the coach. Once the last section (Section 4.2.9.1) adds the estimating and planning of measurements to the continuous integration practice this three level monitoring system covers these SPM tasks.

4.2.9.2.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's continuous integration, with respect to SPM, in the monitoring phase and offer improvements to the process. The SPM shortcomings in this section (Software Development Process, Resources, Effort, Schedule, and Deployment) are not addressed by continuous integration and should be applied to the process to improve it.

The continuous integration practice does not account for monitoring of the software development process (Section 3.3.2) associated with the practice. This practice has the potential to take a significant portion of the teams available effort; therefore it is important for the XP manager to evaluate how the tools, hardware, and software are supporting the continuous integration process and how they can be improved. Section 3.3.2 should be added to the coach and project manager XP process tasks. The continuous integration practice does not address the need to monitor the resources (Section 3.3.4) associated with the practice. As shown for this practice, effort consumption needs to be monitored. Directly related to effort, is how effective are the hardware and software for the practice. Section 3.3.4 should be added to guide the coach and the tracker for the continuous integration practice. Coupled with the monitoring of resources is the team effort (Section 3.3.5). Effort associated with this practice is not monitored or addressed in the XP process. This is a SPM task that should be added to improve the practice. The coach and the tracker should share this responsibility. The planning game (Section 2.5) process handles the overall schedule associated with the continuous integration practice. Unfortunately, the planning game practice only accounts for the continuous integration process once it is set up and running. Section 3.3.6 should be applied to monitor the setup process of the continuous integration practice. Lastly, the deployment monitoring (Section 3.3.8) SPM tasks are not addressed by the continuous integration practice. This practice is where the team practices and deploys the product locally. Thus, it is important to track this practice to ensure smooth deployment at the end of each release.

4.2.9.3 – Continuous Integration—Control & Adjust

4.2.9.3.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Staff, Risk Assessment, Resources, and Measurements) by the XP practice of continuous integration during the control and adjust phase. Those SPM tasks that are not applicable (Effort) to continuous integration during this phase will be identified.

Controlling and adjusting of the staff (Section 3.4.1) is partially covered by the XP coach (Section 2.18.5) and additions to pair programming (Section 4.2.7). The planning game practice (Section 4.2.1) covers all the SPM staffing needs. Risks (Section 3.4.3) associated with the continuous integration practice are identified and methods for avoiding and adjusting risk are defined by the practice (Section 2.13). The XP manager addresses any adjustments of the resources (Section 3.4.4). The coach and the tracker address any adjustments of the measurements (Section 3.4.7).

Controlling and adjusting of the effort (Section 3.4.5) is not applicable to the continuous integration practice. This practice acts as guideline or instruction on the pair programming practice; effort is adjusted in the planning game (Section 4.2.1).

4.2.9.3.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's continuous integration, with respect to SPM, in the control and adjust phase and offer improvements to the process. The SPM shortcomings in this section (Software Development Process, Schedule, and Deployment) are not addressed by continuous integration and should be applied to the process to improve it.

The continuous integration section has shown a need to plan and monitor the setup of this practice. Some overhead and setup are necessary to get this practice up and running. The schedule (Section 3.4.6) for this setup process is not addressed by the practice, and section 3.4.6 should be applied to controlling and adjusting this process. The software development process (Section 3.4.2) is applicable to the continuous integration practice. If changes are required the coach should reference the last team story card associated with continuous integration and write another team story card that can be used in the next iteration or release planning process. Lastly, controlling and adjusting of the deployment (Section 3.4.8) with regards to the continuous integration practice is not addressed by the continuous integration practice. The planning game, pair programming, and continuous integration practices all effect the project's deployment process.

Over many weeks and months continuous integration has the potential to slowly kill the team's velocity. It is important for the coach, the manager, and the tracker to be able to recognize when the team needs make a change to the continuous integration process or refactor the code to decrease build and deployment times. The XP coach and manager closely watch the continuous integration measurements and the team's build and deployment status.

4.2.9.4 – Continuous Integration—Completion

4.2.9.4.1 – Software Project Management Addressed—The XP practice of continuous integration (Section 2.13) covers none of the SPM techniques during its completion phase. Of the uncovered SPM techniques for the completion phase (Section 3.5) none of them are applicable to this XP practice. The planning game and pair programming practices handle all of the SPM completion tasks that arise for continuous integration during the completion phase.

4.2.9.4.2 – Software Project Management Shortcomings & Improvements—None of the SPM techniques defined in section 3.5 should be applied to this phase of the continuous integration practice and there are no recommended improvements.

4.2.10 – 40-Hour Workweek

This section will examine how the XP practice of a 40-hour workweek addresses standard software project management tasks and techniques. The next four sections will examine how XP address the practice of the 40-hour workweek throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases of software project management. Further subsections will identify how the 40-hour workweek practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.10.1 – 40-Hour Workweek—Estimate & Plan

4.2.10.1.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of the 40-hour workweek during the estimate and planning phase. Those SPM tasks that are not applicable (Staff, Software Development Process, Risk Assessment, Resources, Effort, and Deployment) to the 40-hour workweek during this phase will be identified.

The 40-hour workweek is completely undefined in the estimate and planning phase and uses none of the standard project management practices. Most of the SPM tasks are either not applicable to the process or are covered by other XP practices. The 40-hour workweek practice is a supporting guideline for the planning game practice and all of the applicable SPM tasks are handled by the planning game practice (Section 2.5).

4.2.10.1.2 – Software Project Management Shortcomings & Improvements—Most of the SPM techniques defined in section 3.5 are either covered by a parent practice or are not applicable to the 40-hour workweek practice. Only the measurements (Section

3.2.7) SPM tasks are not covered for this XP practice. Most businesses have internal practices in place to cover employee-working hours, if no system is in place the XP team needs to apply a measurement to accomplish this goal. Even though this XP practice is a goal, not a requirement, the management staff has to have a way to track the team's actual hours to compare to the stated goals.

4.2.10.2 – 40-Hour Workweek—Monitor

4.2.10.2.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of the 40-hour workweek during the monitoring phase. Those SPM tasks that are not applicable (Staff, Software Development Process, Risk, Resources, Schedule, Measurements, and Deployment) to the 40-hour workweek during this phase will be identified. All of the monitoring SPM tasks (Section 3.3), except Effort (Section 3.3.5), are applicable to the 40-hour workweek practice.

4.2.10.2.2 – Software Project Management Shortcomings & Improvements—The 40-hour workweek practice does have the necessary system to track effort (Section 3.3.5) but it is not applied in the base XP definition. Part of the planning game practice (Section 4.2.1) and the tracker's (Section 2.18.4) duties should be to monitor the effort or hours worked by the XP team.

4.2.10.3 – 40-Hour Workweek—Control & Adjust

4.2.10.3.1 – Software Project Management Addressed—The XP practice of the 40-hour workweek (Section 2.14) covers none of the SPM techniques during its completion phase. Of the uncovered SPM techniques for the completion phase (Section 3.5) none are applicable to this XP practice. The planning game structure handles any SPM tasks that arise for the controlling and adjusting the 40-hour workweek.

The 40-hour workweek does not directly address a way to control the pressure to work overtime. In an indirect way, by integrating the client into the software development process the XP team manages the clients expectations and tries to deliver the highest priority functionality early in the process.

4.2.10.3.2 – Software Project Management Shortcomings & Improvements—The SPM tasks for controlling and adjusting the 40-Hour workweek are all handled by its parent practice, the planning game. None of the SPM techniques defined in section 3.4 should be applied to this phase of the 40-Hour workweek.

<u>4.2.10.4 – 40-Hour Workweek—Completion</u>

4.2.10.4.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of the 40-hour workweek during the completion phase. Those SPM tasks that are not applicable (Staff, Software Development Process, Risk, Resources, Schedule, Effort, and Deployment) to the 40-hour workweek during this phase will be addressed. In the completion phase, only the measurement (Section 3.5.7) SPM tasks are applicable to the 40-hour workweek.

4.2.10.4.2 – Software Project Management Shortcomings & Improvements—The only SPM task associated with the 40-hour workweek that is applicable and not covered by the practice is the tallying and recording the measurements (Section 3.5.7) of the 40-Hour workweek. The planning game practice should include the completion measurements tasks (Section 3.5.7) for the 40-hour workweek practice into its completion phase (Section 4.2.1.4).

<u>4.2.11 – On-Site Client</u>

This section will examine how the XP practice of the on-site client addresses standard software project management tasks and techniques. The next four sections will examine how XP addresses the on-site client practice throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases of software project management. Further subsections will identify how the on-site client practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

The on-site client practice can be divided into two stages: setup and execution. The setup stage is when the XP management prepares and integrates the client into the XP process. The execution stage is everything after the client is established on-site.

4.2.11.1 – On-Site Client—Estimate & Plan

4.2.11.1.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Software Development Process – setup and execution stages) by the XP practice of the on-site client during the estimate and planning phase. Those SPM tasks that are not applicable (Software Development Process – setup and execution stages) to the on-site client during this phase will also be identified.

The on-site client is one of the main ways the XP process tries to avoid risk in communication and development. Unfortunately this practice only partially addresses the SPM task associated with measurements (Section 3.2.7). The measurements outlined by the XP practice are associated with the client's tasks in the XP process. Measurements for the user stories and acceptance tests are addressed (Section 2.19.1) and staff is assigned to update, monitor and adjust the process (Tracker – section 2.18.4). This XP task minimally covers the SPM tasks associated with the measurement (Section 3.2.7) practice.

The only SPM task that is not applicable to the on-site client in the estimating and planning phase is the software development process (Section 3.2.2).

4.2.11.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's on-site client, with respect to SPM, in the estimate and plan phase and offer improvements to the process. The SPM shortcomings in this section (Staff – setup and execution, Risk Assessment – setup and execution, Resources – setup and execution, Effort – setup and execution, Schedule – setup and execution, and Deployment – setup and execution) are not addressed by the on-site client and should be applied to the process to improve it.

In the setup stage the first SPM task missing in the on-site client practice is the task associated with the staff effort, roles, and development (Section 3.2.1). The on-site client could require more than one person; matching the size (Section 3.2.1.1) of the client team is applicable to this practice and should be covered. The XP process does have rules for choosing an on-site client and lays out a role for them (Section 2.15), but only if there is one client. In practice the XP process may require multiple roles for the on-site client, for example the on-site client could fill three roles:

- Domain specialist – main project contact, usually physically on-site.
- Deployment contact – client the code will be released to.
- Project manager – decision maker who can resolve internal and external conflicts.

The same person could staff all of these three roles, but the roles need to be examined and planned. The application of section 3.2.1.3 should be used to address this issue. In transitioning the client to fulfill their role on the XP team the XP manager has to account for their current job. The client's current job may have a system for staff development (Section 3.2.1.3). To better integrate the client into the XP team and not derail their current development process, the XP manager should coordinate with the client's management and support their professional development if possible (Section 3.2.1.4). Adding professional development (a practice included for staff in standard XP processes) to the on-site client should be considered unrealistic.

In planning for the on-site client, the XP practice lacks the SPM task to define and plan for the risk (Section 3.2.3) that is associated with the practice. The on-site client practice requires resources and flexibility from the XP team. Here are a few potential risks that the XP manager may need to prepare for:

- It may not be possible to gain an on-site client;
- The client may not have the proper domain knowledge;
- The client representative may not possess the authority to guide the team;
- The required resources for the client may be too expensive or impractical to bring to the team;

- The deployment or working environment is only accessible by the client.

Section 3.2.3 should be applied to the on-site client to help minimize some of these potential risks.

Resources for the on-site client can be significant and XP does not address this SPM task (Section 3.2.4). Offices, desks, phones, computers, network support, software, and space need to be planned and costs incurred. The XP manager, the coach, and the client should address SPM tasks associated with resources (Section 3.2.4) before the project kick-off meeting. The easier the integration of the client into the XP team environment, the less disruptive the transition will be for the client and the more useful they will be to the XP team.

Effort associated with XP practices is normally not applicable to the practice, because XP focuses on the utilization of resources over time without adjusting effort. For the on-site client however, the SPM task associated with effort (Section 3.2.5) is applicable because the planning and preparing for client integration into the XP team happens before the planning game process and before the team is actively working on the project. Estimating the effort required will help with scheduling and addresses some of the risks associated with the practice at the beginning of the project. The on-site client practice also does not address the SPM tasks associated with the schedule (Section 3.2.6). The output of the effort assessment should be used to plan and estimate the timeline or schedule for integrating the on-site client practice.

Lastly, the on-site client practice does not address any SPM tasks associated with the deployment process (Section 3.2.8). Any deployment tasks that will involve or require the on-site client's support need to be addressed in this phase. Section 3.2.8 should be applied to cover this SPM shortcoming.

This section shows the need to address risk, resources, effort and scheduling in the estimating and planning phase of the on-site client. These four SPM tasks need to be addressed before the project kick-off meeting with the team. They could be addressed in a typical SPM manner, but it wouldn't fit in the XP methodology. Instead, the XP manager and the coach should address these four tasks using a team story card. This allows the SPM tasks for the on-site client to be integrated into the planning game process.

4.2.11.2 – On-Site Client—Monitor

4.2.11.2.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Staff – execution and Measurements – execution) by the XP practice of the on-site client during the monitoring phase. There are no SPM tasks that are not applicable to the on-site client during this phase.

In the monitoring phase of the on-site client practice the only SPM tasks that are addressed focus on the execution stage of the on-site client practice. The XP manager and the Coach address staff SPM tasks (Section 3.3.1) in the execution stage of the on-site client. Next, the tracker handles the monitoring of all the measurements for the on-site client (Section 2.18.4).

4.2.11.2.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's on-site client, with respect to SPM, in the monitoring phase and offer improvements to the process. The SPM shortcomings in this section (Staff – execution, Software Development Process – execution, Risk – execution, Resources – execution, Effort – execution, Schedule – execution, and Deployment – execution; Setup: All SPM apply) are not addressed by the on-site client and should be applied to the process to improve it.

In the on-site client setup stage the XP management prepares and integrates the client into the XP process. All of the monitoring SPM (Section 3.3) tasks are missing from the setup stage of the on-site client practice and should be applied to this part of the practice.

The previous section (4.2.11.1) recommends adding estimation and planning tasks associated with the staff for the on-site client practice. Thus, the on-site client practice also has a need to monitor the staff (Section 3.3.1) and SPM monitoring tasks should be added to improve the practice. The software development process is applicable to the execution stage of the on-site client and monitoring of the practice (Section3.3.2) should be applied. How the on-site client works in the XP process is well defined (Section 2.15), but having an on-site client may not be possible. The implementation of this practice is important to the success of an XP project and if changes need to be made, the XP manager and the coach need to know early in the project. The XP on-site client practice offers some ideas on how to control and adjust this practice (Section 2.15) but no way to monitor its implementation.

Risks associated with the monitoring phase (Section 3.3.3) are not addressed by the on-site client practice. Risks outlined in the estimating and planning phase need to be monitored in this phase. The on-site client practice does not address this SPM need and section 3.3.3 should be applied. One other risk associated with the on-site client practice is not being able to provide the proper environment for the client. The monitoring of resources (Section 3.3.4) is not addressed by the on-site client practice and should be applied to improve the on-site client's effectiveness.

The setup stage of the on-site client practice is a gating issue for the overall XP practice. Many of the other practices (The Planning Game, User Stories, Testing, etc.) cannot proceed until the on-site client is either integrated into the process or some sort of remote access is defined. Tracking the effort (Section 3.3.5) of this process is not covered by the XP practice. Monitoring of this effort (Section 3.3.5) is a needed addition. The output of the effort monitoring should be used to track the schedule (Section 3.3.6) associated with this XP practice. The on-site client practice does not account for this SPM need and it should be added to improve the practice.

Lastly, how the XP team monitors the deployment (Section 3.3.8) process to the on-site client is not covered by the XP practice. The monitoring of the deployment is addressed by multiple XP practices and deployment-monitoring (Section 3.3.8) tasks should be applied to cover the interaction between the team and the on-site client. How the clients evaluate and provide contacts for access in the upcoming releases needs to be addressed by adding this SPM task (Section 3.3.8).

The monitoring phase highlights seven SPM tasks that are not covered by the on-site customer practice. These SPM tasks (Section 3.3) can be addressed directly and traditionally by the SPM tasks. Adding these SPM tasks to the on-site client practice will not follow the core values of the XP process. In the estimating and planning phase (Section 4.2.11.1) of the on-site client, the needed SPM tasks can be addressed with a team story card. By integrating the on-site clients estimating and planning tasks in to the planning game practice the monitoring of the staff, risks, resources, effort, schedule and deployment are addressed and thus follow the core values of the XP process while addressing the SPM shortcomings of the on-site client.

4.2.11.3 – On-Site Client—Control & Adjust

4.2.11.3.1 – Software Project Management Addressed—This section will identify those SPM tasks that are addressed (Risk – execution partially addressed and Measurement – execution) by the XP practice of the on-site client during the control and adjust phase. Those SPM tasks that are not applicable (Software Development Process – execution) to the on-site client during this phase will also be identified.

The on-site client practice (Section 2.15) partially covers the SPM tasks associated with controlling and adjusting for risk (Section 3.4.3). Risks associated with a client that is not able to be physically on-site are addressed by the practice, but risks associated with the clients themselves are not covered; for example if the client is not completing enough stories, acceptance tests, missing planning game sessions or is ineffective at the position. The tracker (Section 2.18.4) addresses the SPM tasks associated with the measurements (Section 3.4.7) for this phase of the on-site client practice.

The only SPM task that is not applicable to the on-site client during the controlling and adjusting phase is the software development process (Section 3.4.2).

4.2.11.3.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's on-site client, with respect to SPM, in the control and adjust phase and offer improvements to the process. The SPM shortcomings in this section (Staff – execution, Resources – execution, Effort – execution, Schedule –

execution, and Deployment – execution) are not addressed by the on-site client and should be applied to the process to improve it.

The client can be considered part of the XP team and staff. The on-site client practice does not address the need to control and adjust this team member. Section 3.4.1 should be applied to the on-site client to fulfill this SPM requirement. Caution and diplomacy should be used in controlling and adjusting a person that technically is not employed by the XP team. Resource adjustment (Section 3.4.4) is not addressed by the practice. This SPM task should be added to ensure the on-site client has all the tools necessary to work for the XP team and possibly continue to work at their original position. Effort (Section 3.4.5) for acquiring the on-site client and effort required of the on-site client to perform their XP duties are not addressed by the practice. The on-site client may require more than one person, thus implementing the SPM tasks to adjust the on-site client allows the XP team to address a possible project bottleneck. Along with effort goes controlling and adjusting of the schedule (Section 3.4.6). This SPM task is not addressed by the practice and should be used in the setup stage of the on-site client practice.

The last SPM task that is not addressed by the on-site client practice is the controlling and adjusting of the deployment tasks (Section 3.4.8). These SPM tasks should be applied with respect to the release delivery and handover process. Early in an XP process the on-site client or clients will have to take possession of a released product and this phase needs to have the SPM guidelines to handle the practice.

4.2.11.4 – On-Site Client—Completion

4.2.11.4.1 – Software Project Management Addressed—This section will show how the on-site client practice addresses standard project management practices during the completion phase. The on-site client practice addresses none of the SPM tasks (Section 3.5) associated with the completion phase.

4.2.11.4.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's on-site client, with respect to SPM, in the completion phase and offer improvements to the process. The SPM shortcomings in this section (Completion) are not addressed by the on-site client and should be applied to the process to improve it.

All of the SPM completion tasks are unaccounted for and all are applicable to the on-site client practice. To be effective, the on-site client has to integrate quickly into the office and learn the system quickly. Not every client is going to fit into the practice and mistakes are going to be made along the way. That means it is important to addresses these SPM tasks for the acquiring and completion of the project for the on-site client. Staff completion (Section 3.5.8) should be addressed for the setup stage of the on-site client and the overall XP process. In the execution stage, software development process completion (Section 3.5.2) should address how effective they were in creating user stories, acceptance tests, working with the pairs, efficiency in the planning game practice, and how well the on-site client worked in close proximity to the team. The XP process allows for flexibility in these areas and XP mangers should learn from past experience in addressing these SPM tasks (Section 3.5.2). In addition to how well the on-site client practice worked, what risks (Section 3.5.3) were realized and how the team adjusted are not recorded in the on-site client practice. Completion of the risk assessment process should be added to the on-site client practice.

After the on-site client has been integrated into the XP process, the resources used to that end could be freed back to team. Section 3.5.4 should be applied for this purpose during the completion phase of the project. The completion of the schedule (Section 3.5.5) for the setup stage of the on-site client is not covered and should be added. Completion of the setup of the on-site client is important because this first stage is a gating task that must be completed before the main XP process starts. Delays or lost time in this phase can affect the overall project schedule and it is the manager's job to handle this task. The XP team cannot just put in extra effort to adjust for a shorter schedule. When the XP manager addresses the need for completion of the on-site client schedule they also need to address the need to complete the effort (Section 3.5.6) associated with the setup process. Section 3.5.6 only needs to be applied to the completion of the setup stage; other XP practices will cover effort completion in the execution stage. Next, measurements gathered during the setup stage of the on-site client practice need to be completed (Section 3.5.7). Lastly, completion of the deployment (Section 3.5.8) release protocol is not addressed by the on-site client practice. The small releases practice covers the deployment for the project's products; the on-site client's deployment completion covers the system that was set in place by the on-site client for the deployment process.

4.2.12 – Coding Standards

This section will examine how XP handles the coding standards practice with respect to standard software project management tasks and techniques. The next four sections will examine how XP handles the coding standard practice throughout the estimating and planning, monitoring, controlling and adjusting, and completion phases of software project management. Further subsections will identify how the coding standards practice addresses standard project management, what are its project management shortcomings, and how these shortcomings can be improved.

4.2.12.1 – Coding Standards—Estimate & Plan

4.2.12.1.1 – Software Project Management Addressed—There are no SPM tasks that are addressed by the XP practice of coding standards during the estimate and planning phase. Those SPM tasks that are not applicable (Staff, Software Development Process, Risk Assessment, Effort, Measurements, and Deployment) to coding standards during this phase will also be identified.

The coding standard practice addresses none of the SPM tasks laid out in chapter three. From an SPM perspective the coding standard practice should be considered a component or a defined policy of the pair programming practice. All of the SPM tasks, except the schedule and effort, are addressed by the coding standard parent practices (pair programming and the planning game).

4.2.12.1.2 – Software Project Management Shortcomings & Improvements—This section will examine the shortcomings of XP's coding standards, with respect to SPM, in the estimate and plan phase and offer improvements to the process. The SPM shortcomings in this section (Schedule) are not addressed by coding standards and should be applied to the process to improve it.

The only SPM shortcoming associated with the estimating and planning phase of the collective ownership practice is the schedule (Section 3.2.6). Just like the collective ownership practice, the coding standard practice (Section2.16) can be considered a guideline for its parent component, the pair programming practice. Although, neither the pair programming practice nor the coding standard practice defines when or where this guideline will be communicated to the team. This SPM task can be addressed through a team story card (Section 4.2.7). By using the team story card, the coding standard guideline becomes a task that is planned, tracked, monitored, and completed through the planning game practice. The coding standard practice defines how the XP pairs will be expected to perform the coding process. Using the coding standard team story card would be an optimal place to accomplish changing how the code gets written. Some XP books, although undefined, mention the possibility of using CRC (Class Responsibility Collaborator) cards or UML diagrams on story cards. In effect, these are also guidelines that need to be planned and implemented.

4.2.12.2 – Coding Standards—Monitor

4.2.12.2.1 – Software Project Management Addressed—The coding standard practice (Section 2.16) covers SPM tasks associated with the staff, risk, resources, schedule and measurements through the planning game practice (Section 4.21). The coding standard practice is very simple in its concept and implementation and is easily wrapped into the planning game practice. Only the software development processes, effort, and deployment SPM tasks are not applicable to the coding standard practice in the monitoring phase.

4.2.12.2.2 – Software Project Management Shortcomings & Improvements—The monitoring phase of the coding standard practice addresses all of the SPM tasks that are appropriate for the practice and no improvements are recommended.

<u>4.2.12.3 – Coding Standards—Control & Adjust</u>

4.2.12.3.1 – Software Project Management Addressed—The coding standard practice (Section 2.16) covers none of the SPM techniques during its control and adjust phase. This practice is very simple in its concept and implementation. None off the SPM tasks, except the staff (Section 3.4.1) are applicable to this phase of the coding standards practice.

4.2.12.3.2 – Software Project Management Shortcomings & Improvements—In the controlling and adjusting phase of the coding standard practice, section 3.4.1 could be used to direct and guide the staff. Section 3.4.1 should be applied if the team is deadlocked and not moving forward. In summary, the coding standard's SPM shortcomings for the controlling and adjusting phase consist of addressing staffing needs (Section 3.4.1). All of the other SPM techniques are either not applicable or are handled by other XP practices.

<u>4.2.12.4 – Coding Standards—Completion</u>

4.2.12.4.1 – Software Project Management Addressed—The coding standard practice (Section 2.16) covers none of the SPM techniques during it completion phase. Of the uncovered SPM techniques for the completion phase (Section 3.5) none of them are applicable to this XP practice. The planning game and pair programming practices handle any SPM completion tasks that arise for coding standards during the completion phase.

4.2.12.4.2 – Software Project Management Shortcomings & Improvements— None of the SPM techniques defined in section 3.5 should be applied to this phase of the coding standards practice. However, the coding standards practice is one opportunity for the XP team to work through a problem and come to a mutually acceptable solution. In addition, this will be one of the first times for walking through and executing the planning game process. The XP manager needs to archive the results of the coding standards completion phase, while the coach needs to make the results accessible to the team. Then the coach will use the pair programming practice to monitor and adjust their execution.

CHAPTER 5 – CONCLUSION

Throughout the course of this thesis software development theory, the extreme programming process, and the essentials of standard project management have been examined as they apply to software projects. The goal of this examination was to create a management framework for the XP process that provides a robust and accountable development process that better integrates into a wider range of commercial computing projects. The author speculates that the lack of organizational adoption and mainstream integration of the XP process is connected to the lack of standard project management structure that allows for greater control of scheduling, cost, and quality. The XP process offers a rapid development and service-based solution for software development teams. By improving the management practices of the XP development process, greater industrial adoption and additional software development options are available for less structured and dynamic environments.

The main challenge of this thesis was defining both standard project management and the XP process. Each of these subjects is heavily documented with sometimes conflicting information and/or generalities. This large body of information had to be simplified and compressed. The author believes that much of the XP documentation was purposely vague and non-specific. Thus allowing the XP process to appear to attend to SPM tasks when, in fact, very few were defined. By laying out a highly interconnected process (Figure 19) it gives the impression that small changes could unravel the whole practice as the XP authors defined it.

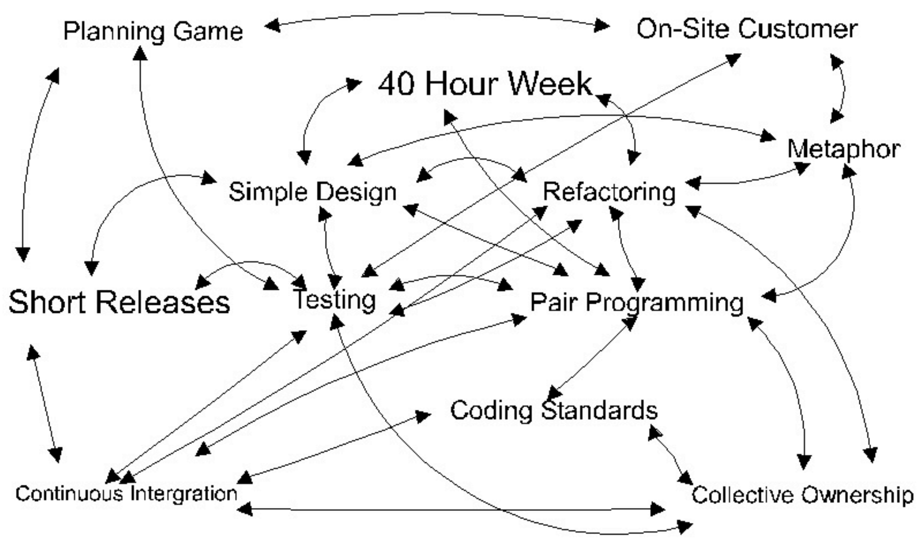


Figure 19 – The interconnectivity of the XP process.

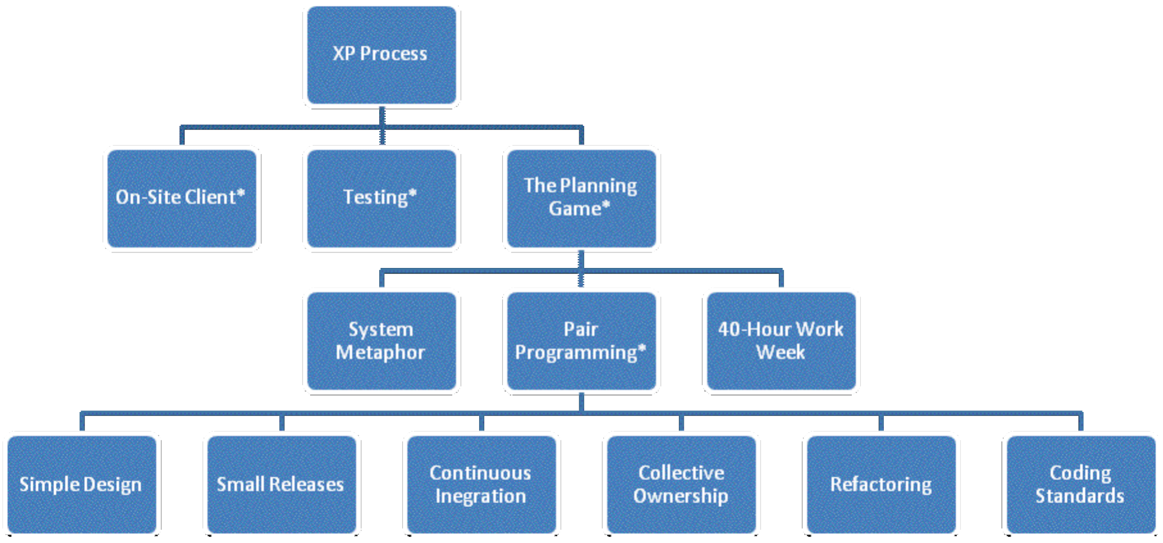


Figure 20 – The management flow of the XP process.

By examining each of the 12 XP practices individually, the author was able to dissect and comprise a better view of the XP process with respect to SPM tasks. Figure 20 shows how the author believes the XP process currently addresses SPM. The XP process does not weight each of the 12 practices equally. The on-site client, testing, the planning game and the pair programming practices are the main XP practices. All of the other eight practices are defined and used as guidelines to support the main practices. Once viewed the in this new structure, the author was able to map how XP address SPM, where SPM tasks were missing, and suggest improvement where possible. Through this mapping of SPM tasks to the XP process the author was able to show:

- The XP practice only addresses 29.52% of the applicable SPM tasks.
- Of the addressed SPM tasks, 79.90% are located in the monitoring and controlling and adjusting phases.
- All of the missing, but applicable, SPM tasks could be added to the XP process while still following the XP core values and maintaining an agile requirement-driven process.
- A management framework could be extracted with the additional SPM tasks added.
- A total of 384 SPM tasks were reviewed with respect to the XP process (8 estimate & plan tasks, 8 monitoring tasks, 8 control & adjust tasks, and 8 completion tasks times 12 practices: 8 * 12 * 4 = 384).
- An additional 127 project management improvements were added to current XP practices.
- On average 31.5 project management improvements were added per project phase (estimating and planning, monitoring, controlling and adjusting, and completion). On average 10.5 project management improvements were added for each of the 12 XP practices.
- A total of 172 SPM tasks are addressed with the additional SPM tasks added by the author.

The author believes that the creation of a management framework for the XP process was successful for this thesis but the process has brought up larger questions about the XP process and SPM.

- Why did the XP authors believe it was necessary to almost completely ignore the estimation and planning phase and the completion phase for SPM tasks?
- Why are there still gaps in software development process when many proven software development practices have been defined for decades?

McConnell states most computer science graduates are "…performing job functions that are traditionally performed by engineers, without the benefit of engineering training" (McConnell 2004, 31). To use a construction metaphor, the XP process appears to be created to give the computer handyman some basic tools they can use in building medium sized commercial buildings. Unfortunately I believe the XP process has failed in this endeavor. By only addressing 29 percent of standard project management tasks, the XP process effectively minimizes software engineering practices and does not move the computer science field toward an engineering science. The XP authors may counter that XP was designed for the "real world environment" and was not designed to be an engineering practice.

The author contends that in not providing SPM tasks to cover their XP practices the XP process is clearly not an engineering practice or a real world practice. The XP practice just doesn't do enough to address the real world planning and completion of tasks. This thesis first strived to identify where SPM tasks were missing in XP and fill them in where applicable to the XP practices. The combination of the XP process and the SPM additions does create a lightweight process that, at a minimum, address SPM best practices.

Future work associated with this thesis could include a comparative study of the effectiveness of the management framework verses the original XP process in a real world setting.

BIBLIOGRAPHY

Beck, Kent. *Extreme Programming Explained.* Boston: Addison Wesley, 2000.

Boehm, Barry. "Spiral Development: Experience, Principals, and Refinements" SEI Joint Program Office, U.S. Department of Defense (2000).

Boehm, Barry. "A Spiral Model of Software Development and Enhancement" *IEEE Computer* 21 no. 5 (1988): 61-72.

Brooks, Frederick P. *The Mythical Man-Month*. Boston: Addison Wesley, 1995.

Callahan, Kevin and Lynne Brooks. *Essentials of Strategic Project Management.* Indianapolis: John Wiley & Sons, Inc, 2004.

Charvat, Jason. *Project Management Methodologies.* Indianapolis: John Wiley & Sons, Inc, NJ, 2003.

Faghih, Farshad. *Software Effort and Schedule Estimation.* Web site: http://www2.enel.ucalgary.ca/People/Smith/619.94/prev689/1997.94/reports/farshad.htm last accessed 12/2007.

Fowler, Martin Refactoring web site: http://www.refactoring.com, (accessed 12, 2007).

Henry, Joel. *Software Project Management.* Boston: Addison Wesley, 2003.

Higuera, Ronald P., Yacov Y. Haimes. *Software Risk Management.* Pittsburg, PA: Software Engineering Institute, Carnegie Mellon University, 1996. CMU/SEI-96-TR-012.

Jeffries, Ron. "What is Extreme Programming?" XP Magazine at Xprogramming.com: http://www.xprogramming.xom/xpmag/whatisxp.htm, 2001.

Jeffries, Ron, Ann Anderson, and Chet Hendrickson. *Extreme Programming Installed.* Boston: Addison Wesley, 2001.

McConnell, Steve. *Professional Software Development.* Boston: Addison Wesley, 2004.

Sommerville, Ian. *Software Engineering.* Harlow, England: Parson Education Limited, 2001. Sixth Edition

Wake, William C. *Extreme Programming Explored.* Boston: Addison Wesley, 2002.

Random House Webster's College Dictionary. New York: Random House, 1996.

APPENDIX A – SOFTWARE PROJECT MANAGEMENT TASKS MAPPED TO XP PRACTICES

The Planning Game:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• Software Development Process (1/2) • Effort • Schedule (1/2)	• Staff • Software Development Process (1/2) • Risk Assessment • Resources • Schedule (1/2) • Measurements • Deployment	• None	• Staff • Software Development Process • Risk Assessment • Resources • Schedule • Measurements • Deployment
Count	*2*	*6*	*0*	*9*
Monitoring Phase	• Software Development Process (1/2) • Resources (1/2) • Schedule	• Staff • Software Development Process (1/2) • Risk Assessment • Resources (1/2)	• Effort • Deployment	• Staff • Software Development Process (1/2) • Risk Assessment • Resources • Measurements
Count	*3*	*3*	*2*	*4*
Control & Adjustment Phase	• Staff • Schedule • Measurements	• Software Development Process • Risk Assessment • Resources • Deployment	• Effort	• Staff • Software Development Process • Risk Assessment • Resources • Deployment
Count	*3*	*4*	*1*	*5*
Completion Phase	• Deployment	• Staff • Software Development Process • Risk Assessment • Resources • Schedule • Effort • Measurements	• None	• Staff • Software Development Process • Risk Assessment • Resources • Schedule • Effort • Measurements
Count	*1*	*7*	*0*	*7*
TOTALS	**9**	**20**	**3**	**25**

Small Releases:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• None	• Staff (3/4) • Risk Assessment • Resources • Effort • Measurements • Deployment	• Staff – Estimation (1/4) • Software Development Process • Schedule • Deployment	• Staff (3/4) • Risk Assessment • Resources • Effort • Measurements • Deployment
Count	*0*	*4.75*	*3.25*	*5.75*
Monitoring Phase	• None	• Staff • Risk Assessment • Resources • Measurements • Deployment	• Staff – Estimation • Software Development Process • Schedule	• Staff • Risk Assessment • Resources • Measurements • Deployment
Count	*0*	*5*	*3*	*5*
Control & Adjustment Phase	• None	•Staff • Risk Assessment • Resources • Measurements • Deployment	• Software Development Process • Effort • Schedule	•Staff • Risk Assessment • Resources • Measurements • Deployment
Count	*0*	*5*	*3*	*5*
Completion Phase	• None	• Staff – Staffing Down • Resources • Deployment	• Staff • Software Development Process • Risk Assessment • Schedule • Effort	• Staff – Staffing Down • Resources • Deployment
Count	*0*	*3.25*	*4.75*	*3.25*
TOTALS	**0**	**18**	**14**	**19**

System Metaphor:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• None	• Staff • Schedule	• Software Development Process • Risk Assessment • Resources • Effort • Measurements • Deployment	• Staff • Schedule • Effort
Count	*0*	*2*	*6*	*3*
Monitoring Phase	• None	• Staff	• Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• Staff
Count	*0*	*1*	*7*	*1*
Control & Adjustment Phase	• None	• Staff	• Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• Staff
Count	*0*	*1*	*7*	*1*
Completion Phase	• None	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• None
Count	*0*	*0*	*8*	*0*
TOTALS	**0**	**4**	**28**	**5**

Simple Design:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• Software Development Process • Schedule	• Risk Assessment • Measurements	•Staff • Resources • Effort • Deployment	• Risk Assessment • Measurements
Count	*2*	*2*	*4*	*2*
Monitoring Phase	• None	• Measurements	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Deployment	•Measurements
Count	*0*	*1*	*7*	*1*
Control & Adjustment Phase	• None	• Measurements	• Staff • Software Development Process • Risk Assessment • Effort • Schedule • Deployment	• Measurements
Count	*0*	*1*	*7*	*1*
Completion Phase	• None	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• None
Count	*0*	*0*	*8*	*0*
TOTALS	**2**	**4**	**26**	**4**

Testing:

	SPM Addressed	SPM Not Addressed	SPM Not Applicable	Added Improvements
Estimating & Planning Phase	• Staff (1/2) • Schedule – Unit Test (1/2) • Measurements	• Risk Assessment • Resources • Schedule (1/2)	• Staff (1/2) • Software Development Process • Effort • Deployment	• Risk Assessment • Resources • Schedule (1/2)
Count	*2*	*2.5*	*3.5*	*2.5*
Monitoring Phase	• Staff • Resources • Schedule • Measurements • Deployment	• Software Development Process • Risk Assessment • Effort	• None	• Software Development Process • Risk Assessment • Effort
Count	*5*	*3*	*0*	*3*
Control & Adjustment Phase	• Staff • Resources • Schedule • Measurements • Deployment	• Risk Assessment	• Software Development Process • Effort	• Risk Assessment
Count	*5*	*1*	*2*	*1*
Completion Phase	• Deployment	• Risk Assessment • Resources • Measurements	• Staff • Software Development Process • Schedule • Effort	• Staff • Risk Assessment • Resources • Measurements
Count	*1*	*3*	*4*	*4*
TOTALS	**13**	**9.5**	**9.5**	**11**

Refactoring:

	SPM Addressed	SPM Not Addressed	SPM Not Applicable	Added Improvements
Estimating & Planning Phase	• None	• Measurements	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Deployment	• Staff • Measurements
Count	*0*	*1*	*7*	*2*
Monitoring Phase	• Measurements	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Deployment	• None
Count	*1*	*0*	*7*	*0*
Control & Adjustment Phase	• Staff* • Schedule*	• Measurements	• Software Development Process • Risk Assessment • Resources • Effort • Schedule	• Measurements
Count	*1*	*1*	*6*	*1*
Completion Phase	• None	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• None
Count	*0*	*0*	*8*	*0*
TOTALS	**2**	**2**	**28**	**3**

*Addressed through other XP practices.

Pair Programming:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• Staff (1/4)	• Staff (1/4) • Risk Assessment • Resources • Schedule • Measurements	• Staff (1/2) • Software Development Process • Effort • Deployment	• Staff • Risk Assessment • Resources • Effort • Schedule • Measurements
Count	*0.25*	*4.25*	*3.5*	*6*
Monitoring Phase	• Staff • Resources • Effort • Schedule • Measurements	• Software Development Process • Risk Assessment	• Staff – Size (1/4) • Deployment	• Software Development Process • Risk Assessment
Count	*4.75*	*2*	*1.25*	*2*
Control & Adjustment Phase	• Staff • Resources • Effort	• Software Development Process • Risk Assessment • Measurements	• Schedule • Deployment	• Software Development Process • Risk Assessment • Measurements
Count	*3*	*3*	*2*	*3*
Completion Phase	• None	• Staff • Risk Assessment • Resources • Effort • Schedule • Measurements	• Software Development Process • Deployment	• Staff • Risk Assessment • Resources • Effort • Schedule • Measurements
Count	*0*	*6*	*2*	*6*
TOTALS	**8**	**15.25**	**8.75**	**17**

Collective Ownership:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• None	• Schedule • Measurements	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Deployment	• Staff • Schedule • Measurements
Count	*0*	*2*	*6*	*3*
Monitoring Phase	• None	• Staff • Measurements	• Software Development Process • Risk Assessment • Resources • Effort • Schedule • Deployment	• Staff • Measurements
Count	*0*	*2*	*6*	*2*
Control & Adjustment Phase	• None	• Staff	• Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• Staff
Count	*0*	*1*	*7*	*1*
Completion Phase	• None	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• None
Count	*0*	*0*	*8*	*0*
TOTALS	**0**	**5**	**27**	**6**

Continuous Integration:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• Risk Assessment	• Resources • Effort • Schedule • Measurements • Deployment	• Staff • Software Development Process	• Resources • Effort • Schedule • Measurements • Deployment
Count	*1*	*5*	*2*	*5*
Monitoring Phase	• Staff • Risk Assessment • Measurements	• Software Development Process • Resources • Effort • Schedule • Deployment	• None	• Software Development Process • Resources • Effort • Schedule • Deployment
Count	*3*	*5*	*0*	*5*
Control & Adjustment Phase	• Staff • Risk Assessment • Resources • Measurements	• Schedule • Deployment	• Software Development Process • Effort	• Schedule • Measurements • Deployment
Count	*4*	*2*	*2*	*3*
Completion Phase	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• None
Count	*0*	*8*	*8*	*0*
TOTALS	**8**	**12**	**12**	**13**

40-Hour Workweek:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• None	• Schedule* • Measurement*	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Deployment	• None
Count	*0*	*2*	*6*	*0*
Monitoring Phase	• None	• Effort	• Staff • Software Development Process • Risk Assessment • Resources • Schedule • Measurements • Deployment	• Effort
Count	*0*	*1*	*7*	*1*
Control & Adjustment Phase	• None	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• None
Count	*0*	*0*	*8*	*0*
Completion Phase	• None	• Measurements	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Deployment	• Measurements
Count	*0*	*1*	*7*	*1*
TOTALS	**0**	**4**	**28**	**2**

*Addressed through other XP practices.

On-Site Client:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• Measurements	• Staff • Risk Assessment • Resources • Effort • Schedule • Deployment	• Software Development Process	• Staff • Risk Assessment • Resources • Effort • Schedule • Deployment
Count	*1*	*6*	*1*	*6*
Monitoring Phase	• Measurements	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Deployment	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Deployment
Count	*1*	*7*	*0*	*7*
Control & Adjustment Phase	• Risk Assessment • Measurements	• Staff • Software Development Process • Resources • Effort • Schedule • Deployment	• Software Development Process	• Staff • Resources • Effort • Schedule • Deployment
Count	*2*	*5*	*1*	*5*
Completion Phase	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• None	• None
Count	*0*	*8*	*0*	*0*
TOTALS	**4**	**26**	**2**	**18**

Coding Standards:

	SPM Addressed	**SPM Not Addressed**	**SPM Not Applicable**	**Added Improvements**
Estimating & Planning Phase	• None	• Schedule	• Staff • Software Development Process • Risk Assessment • Effort • Measurements • Deployment	• Staff • Schedule
Count	*0*	*1*	*7*	*2*
Monitoring Phase	• Staff • Risk Assessment • Resources • Schedule • Measurements	• None	• Software Development Process • Effort • Deployment	• None
Count	*5*	*0*	*3*	*0*
Control & Adjustment Phase	• None	• Staff	• Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• Staff
Count	*0*	*1*	*7*	*1*
Completion Phase	• None	• None	• Staff • Software Development Process • Risk Assessment • Resources • Effort • Schedule • Measurements • Deployment	• Staff
Count	*0*	*0*	*8*	*1*
TOTALS	**5**	**2**	**25**	**4**

Summary:

	Estimating & Planning Phase	**Monitoring Phase**	**Control & Adjustment Phase**	**Completion Phase**	**TOTALS**
Total SPM Tasks Addressed by XP	8.25	22.75	18	2	**51**
Total SPM Tasks Not Addressed by XP	38.5	30	25	28.25	**121.75**
Total SPM Tasks Not Applicable to XP	49.25	43.25	53	65.75	**211.25**
Total Improvements Added	46.75	31	27	22.25	**127**
Total Applicable SPM Tasks	46.75	52.75	43	30.25	**172.75**
% SPM Tasks Addressed by XP	17.65%	43.13%	41.86%	6.61%	**29.52%**
% SPM Tasks Addressed by Adding SPM to XP	100%	100%	100%	100%	