University of Waterloo
Software Engineering

"Implementation of Extreme Programming at CheckFree i-Solutions"

CheckFree i-Solutions
Waterloo, Ontario

Prepared by:

Matthew Stephan
ID xxxxxxxx
mdstepha

1B Software Engineering
September 14, 2003

Matthew Stephan
X
X
X

September 15, 2003

Professor Joanne Atlee
Director of Software Engineering
University of Waterloo
Waterloo, Ontario
N2L 3G1

Dear Professor Atlee:

The following work term report, entitled “Implementation of Extreme Programming at CheckFree i-Solutions”, has been created for CheckFree i-Solutions as required after completing my 1B term. This is my first work term report. The objective of this report is to investigate and scrutinize the execution of Extreme Programming specifically at CheckFree i-Solutions. This report is intended for an individual who has a basic grasp of how software is created, but not, necessarily, any knowledge of Extreme Programming.

CheckFree i-Solutions is the software component of the CheckFree Corporation, and has the responsibility of providing software which enables a business to engage in financial transactions with other businesses or customers. The company is extremely motivated in ensuring that the utilization of online billing continues to escalate.

My specific job at CheckFree i-Solutions was to work with the Business to Business team under the Director of Software Engineering, Mike Toohey. The team’s main responsibility is the creation of a user friendly and efficient web application that allows a particular business to conduct transactions with a number of other businesses via the Internet.

I would like to thank Phil Salmon, Erik Van Der Ahe, Debbie Gutpell, Tony Van Der Valk, Rhonda Henderson, Peter Szabo, and Kevin Thomson for providing me with their prospective on Extreme Programming at CheckFree i-Solutions. I would like to thank Frances Stephan for proofreading this report.

I hereby confirm that I have received no help other than what is mentioned above in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Sincerely,

Matthew Stephan

Contributions

The business to business (B2B) development team at CheckFree i-Solutions consists of four developers, one project leader, and one project manager. The B2B component of CheckFree i-Solutions deals, primarily, with all business to business transactions. As the B2B team, it is the group's responsibility to create a product which allows a specific business to conduct transactions with others businesses online. The group's main goal and ambition is to have the B2B product facilitate transactions that are as efficient and straightforward as possible, for all parties concerned. The team aims to make both the logical components of the application and the graphical user interface (GUI) interrelate coherently. Another key objective for the team is to make the product as generic as possible so that, when a specific company decides to purchase the B2B product, it can be easily customized by both the company in question and the B2B developers. In the instance where a company purchases the B2B software during the development of the B2B product, the team's responsibilities expand to include the development of specific components and amendments agreed upon by the aforementioned company and CheckFree i-Solutions.

The logical components of the B2B application, commonly referred to as the B2B Engine, were completed prior to the commencement of my employment with CheckFree i-Solutions. Thus, the responsibilities accompanied by my job were limited to the development of the B2B GUI component. As a developer on the team, one of my duties was to create the GUI that the transactions would be preformed on, and ensure that the GUI was comprehensible and aesthetically pleasing. This included, but was not limited to, making flow charts indicative of a user's path, deciding where specific components of the GUI belong, and making the user's experience as unproblematic as possible.

After the GUI was created, it was then my task to create code that would allow for the GUI components to interact with the logical components of the B2B application. This code had to be able to respond to a multitude of different possible inputs from the user, and decide what the next logical course of action is. Error handling, efficiency, and correctness were all prevalent concerns of mine when manufacturing the code. In order to verify that the code continues to behave in the manner expected, I was also responsible for creating a large number of automated tests. Since the software must have the ability to be modified by another developer or company, I also had the duty of documenting my code in such manner that would facilitate another developer or company to understand and amend the code as they see fit.

As a member of the B2B team, not only did I have the ability to observe Extreme Programming (XP) in practice, but I was also an active member in the process. The project manager and project leader were continually utilizing XP when allocating the developers time, assigning tasks, scheduling meetings, and planning project cycles. I acquired first-hand experience with key elements of XP including, the discovery and design phase, pair programming, incremental planning, automated testing, short iterations, and a host of others. During my tenure with the B2B team, I attempted to satisfy the XP principles as meticulously as possible by way of using my knowledge of XP I acquired through academic means, and by following the example of the others members of the team. Being a relative newcomer to the software industry, I was able to quickly adapt to the XP process. As my term with the team progressed, both the benefits and shortcomings of XP in relation to our team's goals became quite evident and conspicuous.

CheckFree i-Solutions is one of many companies who are attempting to utilize XP in order to developer software both quickly and efficiently. Since such a key part of having a successful XP project is to use all twelve XP practices simultaneously, a weakness in any one of the practices will have a strongly negative impact. Therefore, it becomes rather imperative to analyze the current implementation of XP at the company, and, more importantly, identify any shortcomings that exist in order to achieve an ideal realization of XP at CheckFree i-Solutions.

Executive Summary

The following report summarizes each of the twelve Extreme Programming practices and scrutinizes the implementation of those practices specifically at CheckFree i-Solutions. Deficiencies in the execution of the practices are then explored and elaborated on. The main purpose of the report is to present CheckFree i-Solutions an in-depth analysis of their implementation of Extreme Programming in order for the company to be able to improve it.

Since CheckFree i-Solutions is a company whose primary function is to develop software and the entire company is implementing the Extreme Programming methodology, the scope of this report touches the majority of the company. Given that developers, project managers, and project leaders are the ones most affected by the practices, this report will be of the most relevance to them.

The main weakness discovered was CheckFree i-Solutions' implementation of pair programming. Although the environment at the company is very cooperative, helpful, and positive, almost no pair programming actually occurs. The main cause of this is the individual developer's resistance to programming in pairs. This inertia is something that has yet to be overcome with developers at the company. A minor issue that may be acting as an obstacle to having a successful pair programming environment is each developer's cubicle is not very supportive of pair programming.

Another prominent flaw was regarding the lack of detail acquired from the planning game practice. The more detail that is associated with each task the easier the developer and project leader can manage time, make estimations, plan important tasks first, and create code easier. It is apparent that some of the tasks defined for a number of projects at the company have the potential to contain much more detail than they are originally bestowed with.

Other notable deficiencies were the lack of coding standards across separate projects within the company, the developers' unwillingness to refactor code until the end of a project cycle, the absence of an actual on-site customer to work with the development team, and the fact that the developers at the company must learn to program with restraint in relation to the simple design practice. Although not as serious as the pair programming and planning game shortcomings, the synergy that exists between the twelve practices is still affected by the aforementioned weaknesses.

Although CheckFree i-Solutions has implemented Extreme Programming admirably, there are still a number of deficiencies which exist that must be addressed. The pair programming and planning game practices contained the most conspicuous shortcomings. Other weaknesses were discovered in a number of the other practices that have a negative effect on the Extreme Programming methodology at the company.

If CheckFree i-Solutions desires an implementation of Extreme Programming that is effective and beneficial as possible then pair programming must become a realization at the company. This can be achieved by assigning tasks in pairs, by encouraging pair programming through seminars and by creating a development environment supportive of pair programming.

The planning game practice at the company must yield more detailed tasks. This could be achieved by allocating a significant amount of time to ensure that each task contains enough detail to satisfy both the customer and the developer. Although time will be expended, the results will be better estimation, and easier time management for developers.

By ensuring that all the practices are performed optimally, CheckFree i-Solutions will be able to use the Extreme Programming methodology much more effectively in the development of software.

List of Figures

List of Tables

1. Introduction

Ever since the introduction of programming languages in the 1960s, software industrialists have been attempting to discover the most efficient methods and practices to be utilized in the manufacturing of software.   As technological advances continue to escalate in both computing and programming, businesses and consumers are demanding a higher quality of software products within a shorter amount of time. The business axiom that time is equal to money still reigns true in today's high paced, competitive business world. Because of that ideology, software companies are sometimes forced to commit to deadlines and, in many instances, the software delivered is not at the level of excellence acceptable for the standards of today. Other factors that can be attributed to the degradation of the overall value of software are poor communication, inadequate human resource management, inaccurate time and cost estimates, and inferior designing and planning of projects.

It is this dissatisfaction with software along with the negative publicity associated with software projects that has generated the need for the Software Engineering discipline. Software Engineers must have an advanced understanding of Computer Science, and be ready to either utilize effective methods of software development or produce new approaches, should the preexisting ones be deemed incapable of enabling a project to reach its goals. Software Engineers must harness all their abilities and training in order to more efficiently manage the four controllable variables that exist in every project: quality, time, scope, and cost [1]. The skill of ensuring that all four variables are managed efficiently, and attempting to find a harmonious equilibrium between them, is one of the defining characteristics of a Software Engineer.

As the field of Software Engineering continues to expand, so has the number of different software development methodologies. The increase in performance of both hardware and software in recent years has only amplified the need for a successful approach. If a team is using a development method that is not well suited for the project in question, there is a high likelihood that the project will fail. Depending on how far along in a project a team is, the cost of failure and the waste of resources can be quite detrimental. Thus, the importance of selecting the correct software strategy becomes a paramount concern for both developers and customer.

Until recently, one of the biggest drawbacks of development approaches was their inability to deal with the exponential relationship between the amount of time the project has been underway and the cost of changing software at that point in time. This is what elevates the Extreme Programming (XP) software development methodology above the rest. Kent Beck, one of the founding fathers of XP, has said "…the exponential rise in the cost of changing software over time can be flattened. If we can flatten the curve, old assumptions about the best way to develop software no longer hold" [2]. Judging by the amount of developers who utilize XP since he first introduced the idea in the late 1990s, he was correct. Figure 1.1 exhibits the fact that as time progresses in an XP project, the cost of change increases minimally. By employing the XP ideology, it is quite clear that the cost of change over time can be greatly reduced.

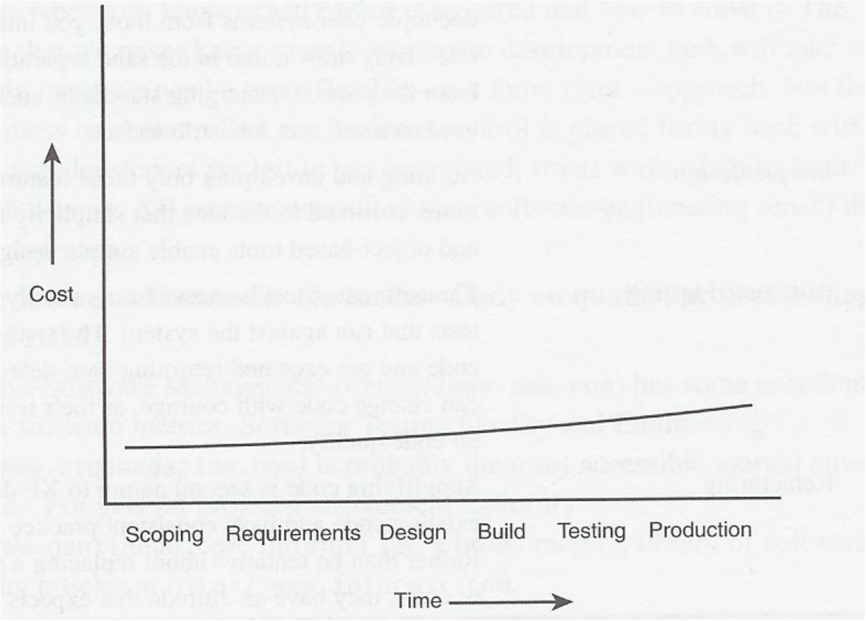


Figure 1.1: Graph of Cost versus Time in an XP project. Taken from [1].

The advantages of keeping the cost of change low over time are quite profound. The main benefit is that it allows developers to keep designs simple. Less anticipation is required for both developer and customer because the expense of not anticipating something becomes much more trivialized. If a feature is not considered essential at a certain moment in time then it can be ignored and, since the cost of developing afterward is relatively the same as it was before, if the feature does become essential it can be implemented economically.

Being devoted to releasing the best quality software product possible, CheckFree i-Solutions has decided that Extreme Programming is the right software development method for them. Employees are instructed in many of the practices and principals relating to Extreme Programming. Project managers and project leaders are encouraged to lead their teams by following the XP model as closely as possibly. Information sessions are held in order to inform developers on how to adhere to the XP methodology and to assist in disposing of old habits that would hinder the success of an XP project.

Understandably, as can be expected when attempting to put into practice a system as involved as XP is, there can be many setbacks and unforeseeable events that occur that can make a methodology difficult to implement flawlessly. Issues such as a person’s reluctance to adapt, economic pressures facing the company, a customer’s specific needs, a lack of sufficient resources, and a plethora of others can all be attributing factors to hindering the implementation of a software development method, such as XP.

This report will present a summary of the twelve practices that are the very essence of making an XP project successful. Furthermore, since “The practices support each other. The weakness of one is covered by the strengths of others” [2], it follows that, in order to investigate the XP practice at CheckFree i-Solutions, an analysis of the effectiveness of each of the twelve practices in the specific context of CheckFree i-Solutions is required. Lastly, recommendations will be made regarding possible improvements to the way in which XP is implemented at CheckFree i-Solutions.

2. Background Information

2.1 The Extreme Programming Methodology

Although the concept of XP is relatively new, the basic activities of XP have been around since people first began creating software. The fundamental activities of any software development project are coding, testing, listening, and designing [1]. What sets XP apart from other software creation methodologies, and the justification for the name Extreme Programming, is that XP takes these activities to the extreme via the twelve practices.

2.2 Practices of Extreme Programming

Similar to the activities of XP, the twelve practices of XP are not novel, but the way in which they are approached is. Each of the twelve practices is performed in such a way that the four basic activities are accomplished as optimally as possible. When examining the twelve practices of XP, it is important to keep in mind that all the practices must be performed well in order for an XP project to be successful. Thus, it is rather crucial that all the practices are executed in order to nullify the shortcomings of another practice. If this is not done, the synergy that exists between the twelve practices will be not as strong as it should be.

2.2.1 40-Hour Week

It is a commonly known truism that the more rested and relaxed a person is, the more focused and driven they are. An employee who is in high spirits and physically healthy will be easier to deal with, more productive, more energetic, and an overall superior member of the team. On the other hand, an employee who is over worked and unduly tired can only have a negative effect on the team. They become volatile, easily frustrated, unproductive, unenergetic, and a poor member of the team. XP deals with this possibility by ensuring that employees are well rested and not made to work excessive amounts of overtime. The practice does not necessarily have an employee work precisely 40 hours a week, as the title would indicate, but simply that they do not perform unwarranted amounts of work. Overtime is acceptable, but overtime for two weeks straight is unacceptable [2]. Also, employees should be allowed to take one long vacation, such as two weeks, and a quantity of off days equal to the amount of days of the long vacation.

2.2.2 Refactoring

One of the most prevalent considerations a programmer takes into account when developing code is the level of simplicity. If the code is already completed, than a developer will attempt to simplify the code as much as possible, while still maintaining the exact functionally of the code before the change. This process is known as refactoring. While most software development methodologies would have refactoring as a single and isolated part of a cycle, the XP strategy has refactoring done on a continual basis. In XP, refactoring is done in order to eliminate any duplication of code, allow modularity, and enable future upgrades to be simpler.

2.2.3 Small Releases

Software that is being shipped to customers should be done in very small releases. That is, it should be able to be released as a functional, although incomplete, product instead of only being ready for shipment biyearly or yearly [3]. Development planning should cover a number of weeks instead of a number of months. This allows planning to be simplified immensely, and allows for developers to use each release as a way to see how well they are doing according to schedule. One of the most paramount benefits of this XP practice, is it allows customers to continually analyze what the developers are doing, and since the customers are not dealing with a finished product, any amendments they suggest to the developers should be less onerous to complete.

2.2.4 Pair Programming

Typically, one of the most neglected practices of XP is pair programming. This is due to a developer's reluctance to program in pairs. Pair programming has two developers situated at one computer. The role of one of the programmers is to consider the most efficient way to implement an operation and then to actual enter the code into the computer. The other programmer's responsibility is to view code from the perspective of possible tests and whether or not this is the correct functionality that was intended for the feature being implemented. This practice is based upon the idea that two minds work better than one, and that code is constantly being scrutinized [1]. Although unpopular with most developers, pair programming has been proven more effective than individual programming. Figure 2.2.4.1 exhibits this increase in efficiency by showing that, in each of the four instances, a larger percent of the tests passed when the code was completed in pairs. It is quite evident that pair programming is a very beneficial practice, especially considering the higher quality of code that is yielded from two people programming rather than one.

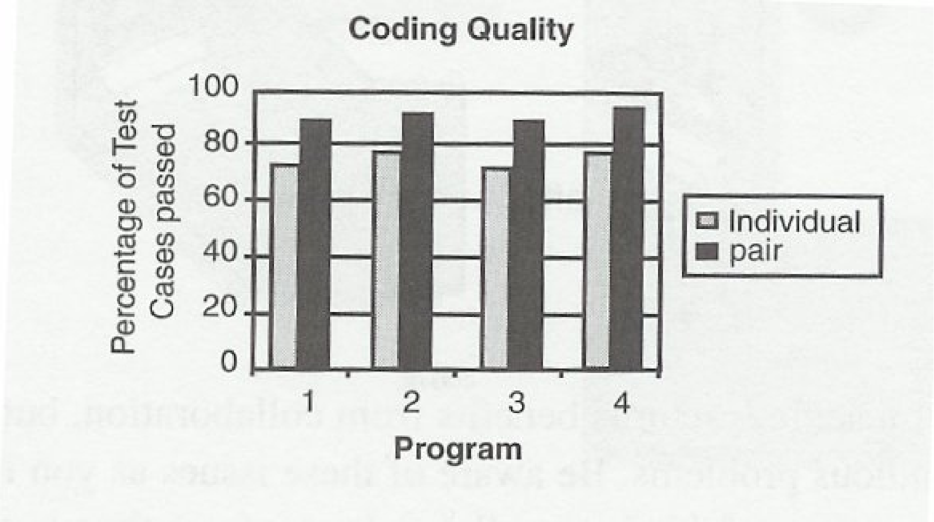


Figure 2.2.4.1: Results from Pair Programming Experiment. Taken from [1].

2.2.5 Simple Design

In order for a software design to be considered simple and correct according to the XP strategy, it must run all the tests, have no duplicate logic or code, be extremely clear about the intention for the developers, and have the fewest number of classes and methods [2]. An extension of this is the idea that if a feature is not required and not requested from the customer then it should not be implemented. This will take up valuable time and there is a high possibility that the feature will never become required nor requested.

2.2.6 Metaphor

Any XP project must have an underlying metaphor that is used to convey to developers how the system works. The concept of metaphor, in this instance, means to label the system as a recognizable idea that has properties or behaviors similar to the system and that can be understood by both the customer and developers. “A good metaphor is a powerful aid in unifying the technical and business teams” [1]. The metaphor utilized will allow for developers to easily understand the system as a whole, and to identify individual components of the system based upon that metaphor. Customers will use the metaphor in order to write up tasks that they want completed. Since the developers understand the metaphor, the developers will be able to clearly ascertain what it is the customer wants. A metaphor may not be required if the system is already understandable to the developers and the customer.

2.2.7 The Planning Game

A common problem that arises in software development is the struggle between which are more important, business issues or technical issues. Developers and customers tend to disagree when it comes to matters regarding these two subjects. The planning game practice ensures that each of the party’s responsibilities are ones to which they are well suited. Table 2.2.7.1 demonstrates that the appropriate responsibilities are given to people who are the best equipped to deal with them. The business people are delegated all the tasks that are important to them and the technical people are assigned the duties which they can accomplish most effectively. Both groups must be as honest and accurate as possible regarding estimations in order for this practice to be successful.

Business People	Technical People
Define scope of the release.	Estimate how long each (task) will take.
Define the order of delivery. Prioritize.	Communicate technical impacts of implementing requirements.
Set dates and times of release.	Break down (feature requests) into tasks and allocate work.

Table 2.2.7.1: Delegation of Responsibilities. Taken from [1].

2.2.8 Testing

One of the most radical practices of XP is testing. Traditionally, programmers would write tests only after development was complete. This XP practice would have developers do the opposite. By writing tests first, programmers are able to create the simplest code possible that would enable the set of tests to pass. Once the tests have all been completed, the code is ready to be added to the entire group's code depository. The set of tests are also added to the depository in order to allow for the tests to be rerun. The task of refactoring is now made much simpler because the only concern for the person doing the refactoring is that the tests pass after the changes are made. The main benefit of testing is that it instills a feeling of confidence in the programmer that their code is behaving in the way they original expected [2].

2.2.9 Coding Standards

In order to maintain consistency across all software components in a system, coding standards should be adopted and enforced by all members of the development team. Some typical standards that can be imposed are naming conventions, formatting, code structure, error handling, and comments [1]. By imposing code standards, the software created will be more fluent, easier to fix, and more coherent.

2.2.10 On-site Customer

One of the ways in which XP deals with the problem of poor communication between developer and customer is the practice of having an on-site customer. This means that an actual end-user of the system, provided by the customer, will be physically situated with the team throughout the project and be a member of the development team [4]. Developers then have the ability to ask questions and seek advice from an actual future user of the system. Although it may be inconvenient for the customer to be separated from their company, the increase in quality due to the customer's participation makes it a fair trade-off.

2.2.11 Continuous Integration

In software development, the term integration refers to the process of amalgamating some of a developer’s recently created code with the system’s current repository of code. The developer then runs all of the system’s tests and ensures that the tests pass in light of the new code that has been merged. The XP practice of continuous integration suggests that developers should do this as frequently as possible. The idea behind this practice is that it becomes extremely clear which code has caused one or more tests to fail because it is the code that has recently been integrated. An effective method of doing continuous integration is to have a computer that is completely dedicated to integration [2].

2.2.12 Collective Ownership

The XP practice of collective ownership suggests that every member of the team is responsible for all code in the project.   A corollary of this is that each member of the team is allowed to change the code in an attempt to improve the quality. The practice of testing makes this idea feasible because, as long as the modified code passes all of the system’s tests, there should be no negative effects from enhancing preexisting code.

3. Analysis: Extreme Programming at CheckFree i-Solutions

Given that the synergy that exists between the twelve practics of XP is so imperative, each practice must be performed as well as possible in order to have a successful implementation of XP. A deficiency in any of the practices negatively affects one or more of the other practices. Although CheckFree i-Solutions has done exceptionally well in adapting the practices, there are a number of shortcomings which become apparent through an analysis of each individual practice.

3.1 40-Hour Week

Developers at CheckFree i-Solutions are given extreme flexibility in deciding the hours they work most effectively. As long as each individual developer programs efficiently for the amount of time allotted for each project task, overtime is rarely needed. Employees are provided with an extremely relaxing break room that assists in ensuring that each developer is working at their peak performance level and not becoming overwhelmed with work. Also, CheckFree i-Solutions strongly follows the XP practice of the 40-hour week in terms of vacation time. Employees are encouraged to take long vacations as well as a number of individual days off. This leaves programmers extremely rested and motivated once they return to work after their vacation time.

3.1.1 Shortcomings

A problem arises in implementing the 40-hour week practice if the planning game practice is not executed well. There are sometimes instances where underestimation has occurred regarding one or more tasks during the planning phase, and employees are then forced to participate in an inordinate amount of overtime hours for several days in a row. When this does occur, employees may become tired as well as being overwhelmed with the amount of hours necessary to complete the project. Normally, an evaluation of the current project would take place, but depending on how far into the project one is, an evaluation may not be viable.

3.2 Refactoring

The automated testing that is in place at CheckFree i-Solutions allows for all programmers to refactor code without having to be concerned about modifying the original functionality of the program. Because of that, many programmers at the company refactor code by taking the latest version of the program from the software repository, modifying the code, and then running the automated tests. After the code passes all tests it then replaces the current version in the software repository.

3.2.1 Shortcomings

Although refactoring is performed at the corporation, it is seldom done unless a developer has some excess time. Even though the key variation regarding refactoring in the XP process is that "…developers will be refactoring during the entire process of development" [1], programmers at CheckFree i-Solutions are not encouraged enough to do refactoring throughout the entire project. Due to deadline dates and commitments, refactoring is often viewed as something that would be nice to have rather than an essential task. Thus, code is not always as simple as possible, making future amendments or enhancements more difficult.

3.3 Small Releases

CheckFree i-Solutions adheres very well to the small releases practice of XP. For every project, cycles of either a two or three week period are scheduled and planned. A number of cycles after a project has commenced, the team is ready to begin shipping small releases of the software. These releases are fully functional components of the overall product and allow the customer to test the product at its current point in development. The small releases of the software are either shipped bi-monthly or monthly.

3.3.1 Shortcomings

On occasion there are components of the small releases that are, in fact, not fully functional components. This can sometimes be caused by dependences between code that is only partly completed and a testable component that requires that code. The project manager then has the responsibility of informing the customer which components can properly be tested. Although informing the customer is not a very onerous task, the XP practice dictates that only fully functional components should be shipped [2].

3.4 Pair Programming

All members of a particular project at CheckFree i-Solutions are situated within the same area of the company's building. This allows for immediate assistance and a significant amount of verbal communication between teammates. In a number of instances, developers will ask for support from one or more developers and that group will then work on the same computer to solve the problem. Another important benefit of the CheckFree i-Solutions work environment is that assistance can, and often does, come from developers outside a particular project.

3.4.1 Shortcomings

Pair programming is undoubtedly CheckFree i-Solutions' biggest pitfall when it comes to the implementation of XP. Pair programming tends to be the most difficult practice to execute, and CheckFree i-Solutions is no exception. Virtually no pair programming is carried out at the company. The main reason for this is the individual developer's resistance the change. CheckFree i-Solutions has very experienced and knowledgeable programmers, some of whom have been professionally developing for twenty years or more. The idea of programming individually is not only fixated in their minds, but is a habit which has proven extremely difficult to break. If the practice of pair programming is to be done properly, then these seasoned developers will have to learn how to program in pairs. Another hindrance of a successful implementation of pair programming at CheckFree i-Solutions is that the developer's individual cubicles are not very supportive of pair programming. Figure A.1 in Appendix A provides a comparison between CheckFree i-Solutions standard cubicle setup and a cubicle setup that supports pair programming. Although it may be onerous to modify the existing system, it is quite apparent that the XP cubicle setup encourages the pair programming mentality.

3.5 Simple Design

CheckFree i-Solutions is very adept at keeping designs simple and efficient. The code created is able to pass all tests, project leaders ensure that there is as little as possible duplicate logic, and the number of classes and methods is kept as small as possible. The class hierarchies used by the company guarantee that the design is kept simple and that amendments and augmentations to the software can be done with minimal difficulty.

3.5.1 Shortcomings

One aspect of the simple design practice that CheckFree i-Solutions has trouble with is the idea of only delivering a feature that is required or requested by the customer. In CheckFree i-Solutions' quest to provide only the highest quality software, project teams sometimes add features to software products that are extremely beneficial to the product but were neither required nor requested by the customer. "To develop with restraint could be one of the hardest things to learn for developers who are new to XP" [1]. The developers at CheckFree i-Solutions tend to deliver a product that has more features than the customer has requested.

3.6 Metaphor

CheckFree i-Solutions is fortunate that the software they develop does not require a complicated metaphor to assist in communication between the developer and the customer. It is extremely likely that each developer has had some experience with financial transactions. When a customer explains that they want a certain object treated in a particular way, the developer easily understands what the customer is referring to.

3.6.1 Shortcomings

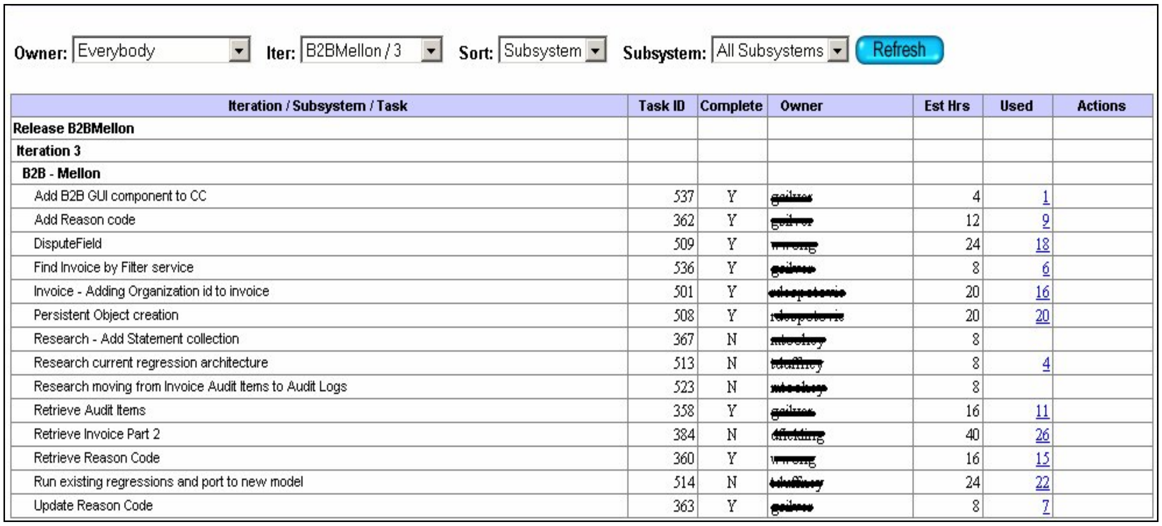
The only possible improvement CheckFree i-Solutions could make regarding the metaphor practice would be to better explain the overall metaphor to newcomers of the company. Although it is rather probable that they will have some understanding of monetary transactions, the metaphor used specifically at CheckFree i-Solutions should be explained in much greater detail. This will allow developers to get a better grasp of the whole software system and enable them to understand customer requests with much less difficulty.

3.7 The Planning Game

The customers involved with CheckFree i-Solutions indicate which days they would like to see the current version of the software. A significant amount of effort by both CheckFree i-Solutions and the customer goes into making a Statement of Work (SOW), which is an extensive document entailing all the tasks required to be completed by both parties. Estimates are made at the beginning of each of the project's cycles, and the estimates continue to become more accurate as developers and managers became more familiar with the project. The work is then divided at the beginning of each cycle in the project and a priority is assigned to each of the individual tasks.

3.7.1 Shortcomings

Figure 3.7.1.1 exhibits a sample of a project tracking tool which allows developers to view and update their status on a particular task via this online tool. It is quite apparent that the tasks are not as divided as they could be, which is CheckFree i-Solutions' biggest shortcoming regarding this practice. The most common complaint from developers is that their tasks are not broken up enough. Having large tasks affects estimations because it is difficult to see where exactly the utilized time went for a particular task. Lastly, the planning game practice has the more important tasks scheduled ahead of the less important tasks, a notion which is sometimes overlooked at CheckFree i-Solutions.



Owner: Everybody Iter: B2BMellon / 3 Sort: Subsystem Subsystem: All Subsystems

Iteration / Subsystem / Task	Task ID	Complete	Owner	Est Hrs	Used	Actions
Release B2BMellon						
Iteration 3						
B2B - Mellon						
Add B2B GUI component to CC	537	Y	[redacted]	4	1	
Add Reason code	362	Y	[redacted]	12	9	
DisputeField	509	Y	[redacted]	24	18	
Find Invoice by Filter service	536	Y	[redacted]	8	6	
Invoice - Adding Organization id to invoice	501	Y	[redacted]	20	16	
Persistent Object creation	508	Y	[redacted]	20	20	
Research - Add Statement collection	367	N	[redacted]	8		
Research current regression architecture	513	N	[redacted]	8	4	
Research moving from Invoice Audit Items to Audit Logs	523	N	[redacted]	8		
Retrieve Audit Items	358	Y	[redacted]	16	11	
Retrieve Invoice Part 2	384	N	[redacted]	40	26	
Retrieve Reason Code	360	Y	[redacted]	16	15	
Run existing regressions and port to new model	514	N	[redacted]	24	22	
Update Reason Code	363	Y	[redacted]	8	7	

Figure 3.7.1.1: Online Project Tracking Tool.

3.8 Testing

The developers at CheckFree i-Solutions are quite meticulous when it comes to testing. Every piece of software developed has an accompanied test that is run at an extremely high frequency. Developers use automated test programs to create tests that they have created. It is made very clear that no software may be deposited into the project's software repository unless all the tests, including its own, are at one hundred percent after the software has been added.

3.8.1 Shortcomings

There are no shortcomings regarding CheckFree i-Solutions' testing practices. In fact, the quality of code at CheckFree i-Solutions is so high that the amount of profit normally accumulated from maintenance has dropped significantly.

3.9 Coding Standards

Within each individual project, coding standards are developed and maintained. If an inconsistency is spotted, the change is made and all tests are run. Standards are sometimes initialized after a project has begun due to a large of number of inconsistencies that are discovered.

3.9.1 Shortcomings

Although code standards are enforced within each individual project, the standards often vary with those of another CheckFree i-Solutions' project. This can sometimes present a problem if a developer is using code from another project in order to model their software based on that code. Also, if standards are initialized in the middle of a project, code that has already been completed may not be updated due to time constraints.

3.10 On-site Customer

Although adding an on-site customer has been deemed unfeasible to implement, CheckFree i-Solutions allows for customers to execute the current version of a product via the Internet. The customer is provided the specific address and password necessary to scrutinize the software. Also, conference calls and continuous communication occurs between both CheckFree i-Solutions and the customer. The company also assigns an employee to act as a virtual customer whose purpose is to test the software whilst emulating the mindset of the customer.

3.10.1 Shortcomings

Despite the fact that CheckFree i-Solutions has attempted vigorously to carry out the on-site customer practice without having an actual customer in the development team, there are a number of deficiencies that are apparent. Although the communication between the two parties is rather fast, feedback is not received as immediately as it would be by having an actual on-site customer. The CheckFree i-Solutions employee that emulates the customer does not always have enough information regarding the deliverables due to lack of preparation for the role.

3.11 Continuous Integration

One of the most notable aspects of the continuous integration practice that CheckFree i-Solutions has adapted is that all developers on the project have access to the current version of the software repository. This allows developers to run all system tests on their own machines, including the code and tests of their newly created software, without actually merging the code.   If the tests pass on the developer's local machine, they are then able to merge their code with the actual version of the product. This is done frequently, as the continuous integration practice recommends. At the end of each day, all the tests are automatically run and a report is sent out to the appropriate members of the project regarding the results of the tests.

3.11.1 Shortcomings

The only evident failing regarding the continuous integration practice at CheckFree i-Solutions is that, although there is a dedicated integration machine which runs tests nightly, not all project teams have there own dedicated integration machine. In order for an integration to occur, developers must add their project to a list of projects to be integrated, and wait a number of hours. Despite the fact that this automatically occurs nightly, developers sometimes require the updated version of the product during the workday. Since some of the project teams do not have their own dedicated machine, time is wasted.

3.12 Collective Ownership

All the developers at CheckFree i-Solutions are encouraged to modify existing code from the software repository given that all the product's tests pass after the amendments have been completed. If a developer discovers that code can be improved, they proceed with the change, run the tests, merge the software with the current version, and inform the owner that they have updated the code.

3.12.1 Shortcomings

Similar to the refactoring practice at CheckFree i-Solutions, this practice is only applied if a developer has extra time during a project cycle. Developers are often so preoccupied with their own responsibilities in a project that, unless inefficiency in a component of the project is very conspicuous, amendments are not done. Although collective ownership is promoted within the project team, improvements are not made nearly as often as they could be.

4. Conclusions

CheckFree i-Solutions has done an exceptional job of implementing XP, especially considering the difficulty associated with attempting to put into practice such a new and radical software development methodology. However, since XP operates best when all of the twelve practices are followed as closely as possibly, each weakness found in implementing the practices should not be taken lightly. Overall, the practices have all been performed rather well, yet there are a number of deficiencies that are rather apparent.

The most notable failing is CheckFree i-Solutions' lack of pair programming. It is just as crucial as the eleven other practices, however it is seldom carried out at the company. Another significant shortcoming is the company's lack of creating detailed tasks when carrying out the planning game practice. Many developers have noticed that the tasks they are assigned are sometimes not as detailed as possible which makes estimations and time management a much more intricate task. The shortcomings in practices such as refactoring, and coding standards stem from the fact that these practices are treated as nonessential tasks and only executed if there is access time in a project cycle. By neglecting these practices, CheckFree i-Solutions is seriously hindering the completely successful implementation of XP.

5. Recommendations

In order for CheckFree i-Solutions to improve their implementation of XP it will be necessary to refine the execution of a number of the XP practices at the company. By enhancing the way that each of the practices is performed, the overall quality of product yielded from the XP development methodology will increase.

To facilitate pair programming at CheckFree i-Solutions, tasks should be assigned to pairs, rather than individuals. This will assist in ensuring that all programming is done in twos. It may appear as if half as many tasks can be assigned because more programmers are being used for individual tasks, but the time that will be saved in testing and refactoring will allow for more tasks to be assigned. Also, developers should be encouraged more to program in pairs via seminars and a rearrangement of cubicles. There will always be some resistance from developers when asked to pair program, but XP will not be as successful unless they learn to adjust.

For the planning game practice, time must be allocated to making project tasks much more detailed. Once a task is created it should be broken up into smaller components in order to allow for better estimation, communication, and time management for developers and customers. Also, the project managers should ensure that the more important tasks are completed first, in order to ensure that each release contains the most crucial elements.

Lastly, practices such as refactoring and coding standards must not be treated as trivial tasks and left till the end of a project cycle. Refactoring and amending code to fit with the standards should be done on a continual basis. Although, an improvement of pair programming should greatly reduce the need for refactoring, it is still an essential part of any XP software project, and must be done with the utmost effort. Coding standards should be established at the beginning of a project via a discussion with the entire project team, and enforced for the duration of the project.

References

[1] S. Baird, *Teach Yourself Extreme Programming in 24 Hours,* Sams Publishing, 2002.

[2] K. Beck, *Extreme Programming Explained,* Addison-Wesley, 1999.

[3] D. Wells, “Make frequent small releases”, http://www.extremeprogramming.org/rules/releaseoften.html (current September 14, 2003).

[4] D. Wells, “The Customer is Always Available”, http://www.extremeprogramming.org/rules/customer.html (current September 14, 2003).

Acknowledgements

I would like to acknowledge Phil Salmon, Erik Van Der Ahe, Debbie Gutpell, Tony Van Der Valk, Rhonda Henderson, Peter Szabo, and Kevin Thomson for providing me information, either through interview or memorandum, regarding their perspectives of Extreme Programming at CheckFree i-Solutions. Their assistance was extremely helpful in allowing me to analyze the implementation of Extreme Programming at the company.

My thanks also go out to Frances Stephan for proofreading this report for grammatical and comprehension errors.

Appendix A: Comparison of Cubicle Setup

The standard cubicle hinders a successful pair programming environment because it becomes very awkward to have another developer at the same workstation. Since the computer is in the middle of the station, when another chair is added to the desk the developer at the end of the table will not have a clear view of the computer screen. Also, the developer at the end of the table will feel uninvolved with the development of the software because their proximity to the actual computer is quite far.

The pair programming supportive cubicle places each developer at equal lengths away from the computer. Each developer has easy access to the computer and is now able to work much more effectively as a team.

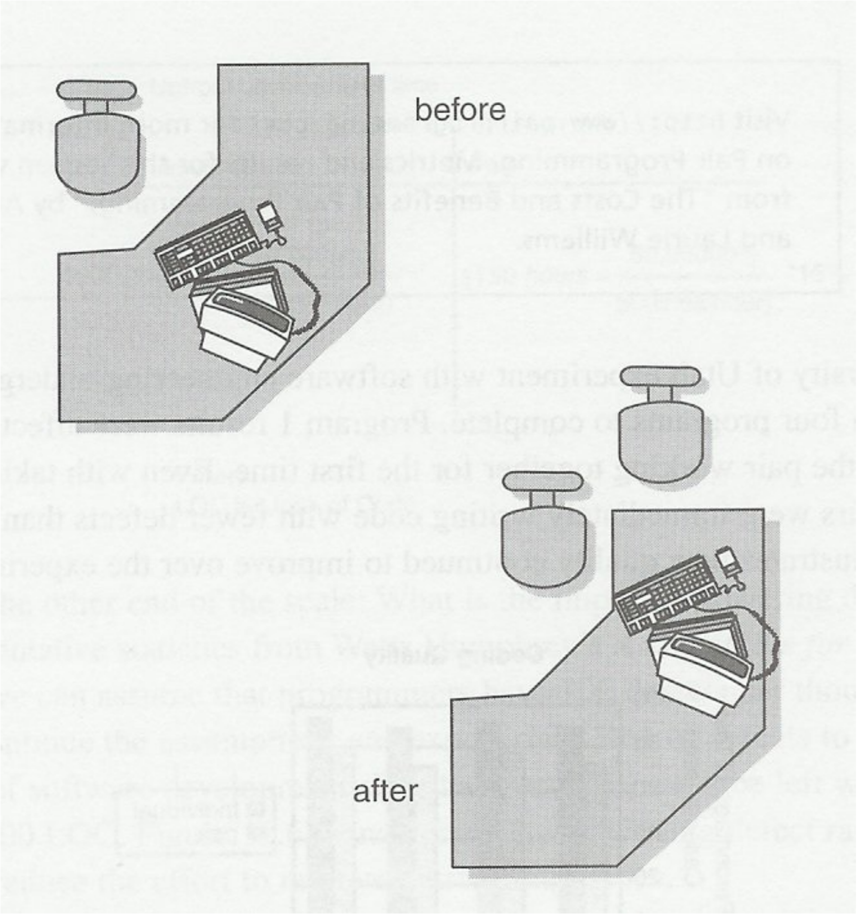


Figure A.1: Comparison of Cubicle Setups. Taken from [1].

**The before desk is the current cubicle setup used at CheckFree i-Solutions.
The after desk is the pair programming supportive cubicle setup.**