University of Waterloo
Software Engineering

“Analysis of Rational ClearCase as an Extreme Programming Tool”

CIBC Mortgages Inc.
Toronto, Ontario

Created By:

Matthew Stephan
ID xxxxxxx
mdstepha

2B Software Engineering
May 10, 2004

Matthew Stephan
X
X
X

May 10, 2004

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

Dear Professor Atlee:

The following report, entitled “Analysis of Rational ClearCase as an Extreme Programming Tool”, has been prepared for CIBC Mortgages Inc. as my 2A work report. This is the second work report that I have completed. The purpose of the report is to analyze the effectiveness of the software tool Rational ClearCase when it is used in conjunction with the Extreme Programming development methodology. The report was written with the intention of allowing people who possess a fundamental understanding of software development to be easily able to grasp and retain the concepts and ideas presented in this report. Sufficient information will be provided regarding Rational ClearCase and Extreme Programming, thus ensuring that the audience indicated will not find the material overwhelming.

CIBC Mortgages Inc. provides mortgages and money lending to a significant amount of consumers in order to allow these people to purchase homes without having all the required capital necessary to obtain a house.

My placement at the company was with the Intranet/Internet team, and I was under the direct supervision of Mirek Zyskowski. As part of the CIBC Mortgages Inc. Information Technology department, my team was responsible for amending and maintaining the Intranet and Internet sites for each of the individual departments that belong to CIBC Mortgages Inc.

I would like to acknowledge Mirek Zyskowski of CIBC Mortgages Inc. for furthering my understanding of Rational ClearCase and its applicability to developing software efficiently. I would also like to thank Frankie Stephan for the assistance she provided via 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
ID xxxxxxx

Contributions

The Information Technology department at CIBC Mortgages Inc. (CMI) consists of approximately two hundred employees. This department is comprised of numerous different teams, each run by the team's respective manager or project leader. Some teams have a very active role in selling mortgages to customers via creating and maintaining the software that the CMI mortgage sellers use when acquiring a customer, while other teams perform a more passive function by means of providing internal support to other departments and ensuring that all hardware and software is operating within normal parameters.

My duties at CMI entailed me joining one of the more passively involved teams, the Intranet/Internet team. This team consisted of three members: my supervisor, another university level student, and myself. The Intranet/Internet team was responsible for ensuring that both the Intranet and Internet Web sites for each of the departments that comprise CMI were updated and maintained as frequently as needed. The Intranet sites were the Web pages that could be viewed only by employees of CMI, while the Internet sites were the Web pages that were visible to everyone. The team had to respond to requests from different departments regarding a myriad of different issues, ranging from adding an application, amending documents, or modifying the aesthetical appearance of a site. The overall team goal was to respond quickly and efficiently to any problems or updates that the other departments deemed necessary for our attention. Since the other departments that encompass CMI were, in fact, the team's customers, a related goal was to ensure that our customers were dealt with pleasantly and were encouraged to maintain an open line of communication regarding each update that was being performed.

In the past, the Intranet/Internet team at CMI had never utilized university students, thus forcing my supervisor to assume full and sole responsibility of both the Intranet and Internet sites. This implied that he was forced to perform his normal development tasks while performing the Intranet/Internet team tasks as well. This was deemed too overwhelming for one person to cope with and it was decided that it would be prudent to bring in university students to join the team. My responsibilities were to assume a significant amount of the related tasks that the team received in order to help alleviate the pressure that was plaguing my supervisor. I performed a considerable amount of updates daily, including creating and modifying interactive Web forms, updating Web pages for both aesthetics and content, developing and maintaining the software required to run the pages, and a multitude of other tasks.

Since multiple members of the CMI Intranet/Internet team work on the same files at different times throughout the course of a project or a task, it is quite logical that the team would utilize some sort of tool to ensure that the correct version of the files are being worked on and that synchronicity is achieved throughout the project. The particular tool that the team, and many other departments within CMI Information Technology, used to satisfy these concerns was Rational ClearCase (or ClearCase). An issue that arose when a tool such as ClearCase is used is the question of whether or not the tool is compatible with the specific software development methodology that the team chooses to employ. The methodology that the Intranet/Internet team utilized was the Extreme Programming (XP) methodology. Working with both ClearCase and XP it became quite apparent to me that there were both benefits and shortcomings that arose when the two were used in conjunction with each other, and thus spawned the basis for this report.

The majority of departments in the CMI Information Technology group utilize ClearCase for the purpose of ensuring that consistency is maintained regarding files, and most of the departments utilize some, if not all, of the XP practices. Thus, the work and analysis that is accomplished in this report will allow the company to re-evaluate the way that they are using ClearCase, and subsequently, make the changes necessary to guarantee that ClearCase is used as efficiently and correctly as possible when working in conjunction with XP.

Executive Summary

The following report scrutinizes the utilization of ClearCase when it is used in conjunction with the Extreme Programming (XP) software development methodology. Background information is provided on XP and ClearCase in order to facilitate a better understanding of the analysis that is provided. Once that has been accomplished, the XP practices that are affected by ClearCase are examined and the benefits and shortcomings of each are considered. Conclusions are then drawn regarding ClearCase's ability to act as an XP tool. Lastly, recommendations are made regarding the way that ClearCase's inadequacies concerning XP can be diminished.

The principal objective of this report is to provide developers and project managers a basis for determining whether or not they should use ClearCase in their XP projects. Although the benefits overshadow the shortcomings, the developers or the managers may determine that the deficiencies are too severe to compensate for. Since the majority of departments situated in CIBC Mortgages Inc. use ClearCase and XP, the scope of this report reaches the bulk of the company. Project managers and lead developers will find the report the most relevant, because they are the ones who oversee the execution of the XP practices. The scope of the report only touches on the four XP practices that are affected by the use of ClearCase.

For each of the four practices analyzed, the advantage of using ClearCase pertaining to those practices is quite evident. The testing practice is facilitated by allowing developers the ability to easily acquire the most recent versions of the tests, as well as use the ClearCase version history feature to view recent changes that may have affected test results. Refactoring using ClearCase is efficient because ClearCase has a

merge feature that allows multiple users to simultaneously work on a file and be able to merge it after both developers are finished making their amendments. The continuous integration XP practice is practically performed automatically by ClearCase, which is a significant benefit. The collective ownership practice is also performed inherently, because ClearCase is based upon allowing multiple users to work on the same files.

Notable weaknesses that ClearCase has for an XP project include the need to constantly retrieve the newest versions of the tests, the way that many teams neglect application and user testing due to lack of a integration machine, the requirement of updating developer's projects each time refactoring is performed, and the fact that all developers are able to modify the system's tests. Despite these failings, ClearCase still proves to be an effective and useful XP tool.

The respective recommendations for dealing with the aforementioned weakness is to have tests modified by specific people at certain times only, make an integration machine mandatory for all development teams at CMI, have refactoring performed only in a refactor phase of a project or during emergencies, and have restrictions placed on the modification of the system's tests.

Since CMI already uses ClearCase and the XP development methodology, they must attempt to rectify the shortcomings as hastily and correctly as possible if they wish to minimize the problems that can occur when the two are used in conjunction.

Table of Contents

List of Figures

List of Tables

1. Introduction

Understandably, when various developers are working together on common files, it is a logical progression to realize that many conflicts can occur regarding file synchronicity and version problems. Without protection against such conflicts, developers can accidentally edit the wrong version of a file, amend or delete a file while another programmer is using that file, or have an application/program that functions differently than another developer due to an inconsistency of files between the two members of the team. These problems make developing, managing, and testing software significantly more onerous than it should be.

Recognizing that these issues needed to be addressed, makeshift solutions were created and used by many developing teams. An example of such a solution was to have one or more members, of the team acting as the project "librarian". This person distributed the software that was to be amended to other team members, meticulously took down which team members were working on which files, and recorded the new versions as soon as people had submitted their changes back to the librarian. [4] This process was flawed and proved to be inefficient and impractical, especially considering the demand of the fast-paced environment that is frequently associated with software development. As the complexity and magnitude of software projects continued to increase so did the need for an effective way to deal with these problems. This instigated the creation of a new genre of software tools, known as Software Configuration Management (SCM) Tools.

An SCM tool combats problems associated with maintaining file consistency and version regularity. It automates a substantial amount of the processes that are required to perform effective SCM. Projects are

becoming so immense and intricate that "an SCM tool is an essential part of every software engineer's tool kit today."[4] Table 1 displays the basic features inherent in any SCM tool. These features are the fundamental and defining criteria for a SCM tool to be successful and are the main focus of early SCM tools, such as the source code control system (SCCS) and the revision control system (RCS). [4] Both the SCCS and the RCS implement the basic version-control features shown in Table 1, thus eliminating the need for having a member of the team act as a librarian. For many of the elaborate and substantial projects that exist today, the SCM necessary to have a project run as smoothly as possible requires an SCM tool that is more functional than the SCCS or the RCS. More advanced SCM tools must allow larger project teams to be able to deal with the new and multifaceted problems that face developers today.

Basic Version-Control Feature
To maintain a library or repository of files
To create and store multiple versions of files
To provide a mechanism for locking files
To identify collections of files
To extract/retrieve versions of files from the file repository

Table 1: Basic Features of a SCM Tool. Taken from [4]

One such tool that has become exceedingly popular among software development teams, including the teams at CIBC Mortgages Inc. (CMI), is the Rational Software product entitled ClearCase. ClearCase is a SCM tool that not only provides the basic functions for file consistency, but also is a product that is able to work in conjunction with project management tools and third-party development utilities. [3] ClearCase, in conjunction with Rational Software's ClearQuest, is a major

component of the unified change management (UCM) method, which entails bringing together all the tasks that relate to changing software and tracking the changes of a project. [4] The utilization of ClearCase is becoming extremely wide spread. Developers and managers continue to ascertain whether or not the software development strategy that they are using is compatible with ClearCase.

A development methodology that is often used concurrently with ClearCase is Extreme Programming (XP). XP, original conceived of by Kent Beck, is one of the fastest growing and new-age software development approaches that are in use today. [1] The fundamental ideology behind XP is that it takes the 4 software development activities: coding, testing, listening, and designing, and attempts to perform these activities constantly and immensely. [2] XP utilizes 12 practices in order to accomplish its goal of deliver software both proficiently and swiftly.

SCM tools are not applicable to all of the 12 practices that comprise XP, but since such a high emphasize is placed on having the practices work in harmony, it is imperative that the SCM tool that is being used ensures that the related practices are being executed effectively. As was noted in CMI's use of ClearCase and XP, this is not always the case. Variances in the way ClearCase is utilized can cause practices to be neglected or hindered and can seriously damage the effectiveness of a team.

This report will provide a summary of ClearCase, and XP. An analysis of ClearCase's effectiveness as an XP tool will be presented and elaborated upon. Conclusions will be made concerning the analysis, and lastly, recommendations will be stated that will allow ClearCase to be used optimally as an XP device.

2. Background Information

2.1 Rational ClearCase

ClearCase is a SCM tool that allows developers to make numerous amendments to files situated within a project without having to worry about problems like file synchronization and file version inconsistencies. It is compatible with a significant amount of systems, including, the two industry standards, Microsoft Windows and Unix. It has proven effective in a myriad of different types of software projects including application software, Web projects, financial applications, and many more. [4]

Having the ClearCase Administrator properly setup ClearCase is paramount to the success of any ClearCase implementation. The administrator tends to be a person who has a thorough comprehension of the system architecture and the overall design structure of the project. The setup of ClearCase entails defining and organizing the structure of a ClearCase view. A view can be described as a layout of all the directories and files situated in a project. [3] It is analogous to taking a photograph of the project's data and allowing this information to be perused and retrieved by developers.

Once the view is setup, the true ClearCase process is ready to commence. Table 2 provides a summarization of the ClearCase progression as it applies to a standard development project. Steps 1 and 2 of the table are preliminary steps that are done by the ClearCase Administrator and the Project Manager, respectively. The developers and the ClearCase Administrator accomplish the remaining steps. This set of steps, steps 3 to 8, can often be done out of order or repeatedly if the need to do so arises.

#	Description of Step
1.	Software files and directories are organized into versioned components.
2.	Project managers create projects and assign project teams to work on these components.
3.	Developers make changes to components, files, and directories based on assigned activities.
4.	New file and directory versions are collected during development and associated with activities.
5.	Activities and their associated change sets are delivered and integrated in a shared project integration area.
6.	New component baselines are created, tested, and promoted.
7.	Component baselines are assembled into a system.
8.	Systems are tested and released.

Table 2: Underlying Steps in the ClearCase Process, Acquired from [4]

One of the most advantageous and noteworthy features of ClearCase is that, after the ClearCase application has properly been setup and configured, it is extremely easy to use and is also quite helpful to developers. As seen in Figure 1, after a developer joins a project, they enter a straightforward and efficient cyclic process. The Make Changes element of the cycle has the developer "check out", or reserve, the file or files that they wish to work on, and then institute the amendments they desire. To ensure file synchronization, ClearCase ensures that only the most recent version of the file is accessed and modified and that the file is not checked out or reserved by any other developer on the team. That is, ClearCase informs the user that they must update the local version of the view that is on their machines, also known as a workspace. This updating is what is done in the Update Workspace component of the cycle. Whenever a developer decides it is time to update their workspace, ClearCase determines the variances between the files and updates the user's workspace accordingly by adding, changing, or removing files that

comprise the view. The Deliver Changes part of the cycle has a developer finish their alterations to a file or a directory, and then "check in", or return, their modification in order for the view to be updated. That element is now available for all the other members of the team to amend, assuming they first update their view. Developers are able to continually navigate through this cycle without being concerned with the main problems associated with SCM.

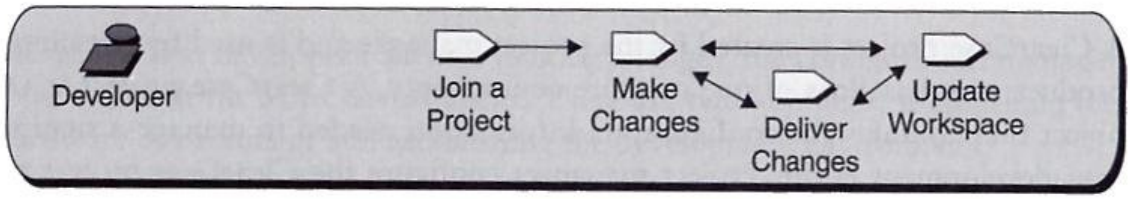

Figure 1: The ClearCase Development Cycle. Taken from [4]

2.2 Extreme Programming

One software development approach that has proved successful in a multitude of different projects, including the ones conducted at CMI, is XP. A relatively new and radical methodology, XP's main philosophy is to take the 4 basic software creation activities and perform them to the extreme. [2] These activities are coding, testing, listening, and designing. Coding involves the actual creation of code that is used when manufacturing a program. This activity is the constructing and documenting of the code required to have an application function. Testing entails producing and executing the tests necessary to ensure that a program is functioning as expected. The listening activity necessitates having the developers or the developer's manager listen and communicate with the business side of a project in order to guarantee that the program fits well with the customer's needs. The final activity is designing which requires the project manager and lead developer to get

together and work out how the project's resources can be effectively and practically managed. [1]

In order to perform the 4 basic development activities as intensely as possible, XP exploits 12 practices that are to be performed while developing software. Table 3 in Appendix A contains a brief description of each of the 12 practices. These practices are the true crux of XP and are the justification that the methodology is considered so revolutionary. It is crucial that these practices are followed as closely as possible because, as exhibited in Figure X located in Appendix A, "the practices support each other. The weakness of one is covered by the strengths of others." [2] This implies that if one of the practices is neglected then all of the other practices will be hindered in terms of success. This interdependency between the practices is what sets XP apart from all other development methodologies and is what causes XP to be especially successful, when executed correctly. [1]

Since ClearCase only relates to and affects a subset of the XP practices, it is prudent to elaborate on only those practices. Although Appendix A contains a summarization of all 12 practices, having a better understanding of the practices that are affected by ClearCase will facilitate the comprehension of the analysis that is being presented.

2.2.1 Testing

The testing practice in XP entails having a developer create automated tests for each of the software components before the code is actually written. The tests do not cover every aspect of an application; rather the tests envelop "only production methods that could possibly break." [2] Once the tests are created, any changes made to a project component must pass these tests. If amendments cause the software to

fail the test, then the modifications are not incorporated in the main software repository. By creating the tests before the software is created, developers are easily able to identify where and when problems have arisen, and are capable of attaining a new level of confidence regarding the functionality and correctness of their code.

2.2.2 Refactoring

The XP practice of refactoring involves having a developer determine if there is a way to simplify an existing feature in a program that will still allow the tests for that program to run successfully. [2] Since each of the application's tests already exists, refactoring can be performed without being concerned about unduly modifying a system's functionality. Refactoring is performed after a feature has been added, or if it has become necessary to duplicate code, thus implying that simplification is warranted. [1] This practice ensures that the project's code is both simple and functional.

2.2.3 Continuous Integration

The continuous integration process necessitates the amalgamation of all of the recent code amendments to a dedicated integration machine at least once a day. This machine hosts the most recent version of the entire application and allows user and efficiency testing to be performed. Each member of the team integrates their recent changes into the machine, and then guarantees that all of the system's tests pass. [1] Since only one person integrates their changes at a time, any failures in the system's tests are conspicuously the fault of the person currently integrating, especially since the person who integrated previous left the system's tests at 100%. [2] Continuous integration ensures that all of the project's components successfully interact with each other.

2.2.4 Collective Ownership

Collective ownership is as much an idea as it is a practice. What it entails is having every member on the team be equally responsible for every component of the project. In the previous software development methodologies ownership of code was the sole responsibility of the member who created it. If other members of the team saw fit to change that code, they were forced to contact the owner and submit a request. [2] The XP concept of ownership allows any member of the team to simplify code if an opportunity to do so presents itself. Since all members of the team have at least a basic understanding of every component in the system, they should be able to ascertain whether an amendment is warranted. Since tests must run successfully after the modifications to the code, the other members of the team do not have to be concerned with the possibility of a change in functionality.

3. Analysis: ClearCase as an XP Tool

Although ClearCase is often used in conjunction with the XP methodology, the effect that ClearCase has on the individual practices is rarely considered due to the popularity and availability of the product. It is quite probable that the structure and processes involved with ClearCase may, in fact, hinder one or more of the practices. Thus, the appropriate practices are analyzed below.

3.1 Testing

ClearCase facilitates the XP practice of testing by having programmers add the component and the component's respective tests into the software repository. As exhibited in Figure 2, testing occurs on a continual basis via having the developer "check out" the component that they desire with ClearCase, then performing amendments on the file and ensuring that those amendments pass all tests, and lastly, placing the file back into the file repository through ClearCase. As seen in the figure, the aforementioned cycle does not begin until the tests and the code itself has been positioned inside the file repository.

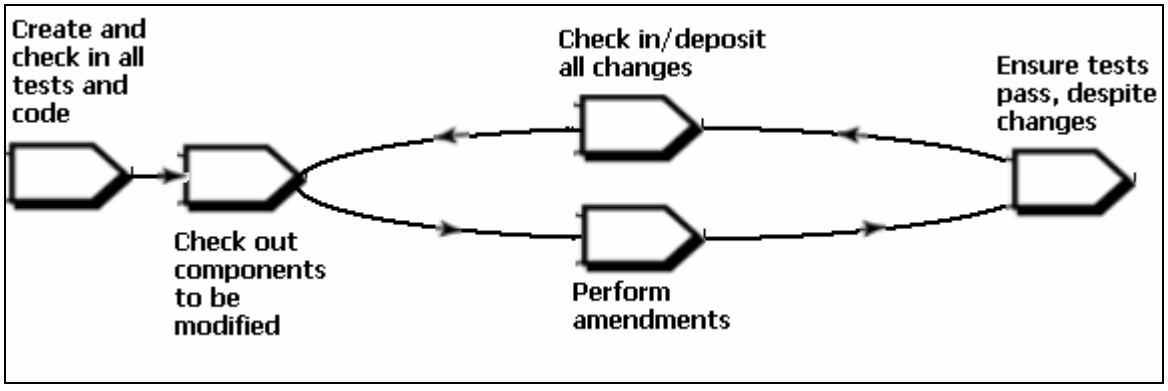

Figure 2: XP Testing Phase being accomplished via ClearCase

3.1.1 Benefits

Once the initial code and tests are placed in the repository, testing using ClearCase becomes extremely rudimentary. ClearCase facilitates XP testing by allowing developers to easily obtain the latest versions of the tests and code frequently without having to be apprehensive about having an incorrect version of the files. If ClearCase was not being utilized, it is very feasible that two developers would be working on the same component at the same time.

Another advantage of ClearCase in regards to the XP testing practice is the version history that ClearCase provides. The version history feature allows a member of a ClearCase project to view the list of people who have modified an individual file. It also shows the respective changes that each person made. This proves advantageous in a number of situations. The first situation is when a developer decides that the functionality of a program should change, and thus, changes the tests and code for that component accordingly. If the original creator of that code views the newly modified code and tests, they may not be aware of what instigated the change in functionality. They could then contact the team member who modified the code and ascertain what the justification for the amendment was. The second situation that the version history feature proves worthwhile is in the rare instance that a developer has modified code without certifying that all the tests for that component have passed. In this instance, that developer can be tracked down and informed of their mistake in order for them to rectify it.

A more abstract benefit of ClearCase concerning the XP testing practice, is that it provides developers a clear-cut way of determining the appropriate procedure for performing their code changes. A project that does not utilize ClearCase may not run their tests consistently or at the

appropriate time. However, with ClearCase, developers are well aware that all of the application's tests must be run and successfully passed before the changes are checked in to the file repository.

3.1.2 Shortcomings

The most notable failing that ClearCase has pertaining to the XP testing phase is the requirement of all the developers to continually update the files situated in their view in order to guarantee that the tests they have are consistent with the other members of the team. This implies that every time a programmer needs to run their modified code against the system's tests, they need to retrieve the latest version of those tests and all components relating to the modified component. While this may appear trivial at first glance, for large projects that contain 10 team members and an inordinate amount of code, this flaw becomes increasingly detrimental. Depending on the size of a view, ClearCase updates can take as long as 20 minutes. [4] For developers who run multiple tests a day, the effect of this can seriously hinder productivity.

3.2 Refactoring

The XP practice of refactoring is rather straightforward when being accomplished using ClearCase. Developers have the ability to check out current versions of any of the files located in the project. Once this has been completed, developers are then able to simplify and optimize the components as they see fit, assuming the amendments made do not cause the code to fail the tests. Since the functionality of the application is not affected by refactoring, members of the team can continually check out, run tests, and check in files without being concerned with the repercussions.

3.2.1 Benefits

One benefit of using ClearCase for refactoring in an XP project is that ClearCase has a merge feature. This feature allows a developer to work on a component of a project that is already checked out by another member of the team. This is quite effective in situations where different team members are working on different elements of a single project component. A prominent example is if one programmer is documenting some code while another programmer is appending functionality to the program. Although the person changing the actual code would have the file checked out, the developer who is performing the documentation will be able to run a merge and add their documentation to the newest version of the file.

Another advantage of using ClearCase for refactoring is ClearCase's ability to view the past history of a single file. This enables developers to see the entire evolution that a file has gone through; thus, facilitating the ability for developers to ascertain what refactoring steps may still be required. Having the file history also helps developers find out what changes, if any, were made to a file they may have originally created.

3.2.2 Shortcomings

One shortcoming that exists when using ClearCase in conjunction with the XP refactoring practice is that ClearCase maintains software consistency via only allowing one user to truly modify a project element at a time. While the merging feature is more than an acceptable solution for minor or trivial changes, complex and intricate amendments to a component need to be accomplished by separate developers at different times. This may not be a problem in the earlier stages of production, due to the multitude of different tasks that a developer may be responsible

for early on, but if it is near the end of the development cycle and a single component is not functioning correctly, then the need to have only one developer amend a component at one time becomes a considerable hindrance to efficiency.

A more conspicuous inadequacy regarding ClearCase and the XP refactoring process is that there is a direct correlation between the quantity of project components that are modified and the number of updates that a developer must perform on their view. Although updates must be executed irrespective of whether refactoring is being accomplished, refactoring still results in the modification of a system. Since a change has been made to the application, a developer will have to update their view in an instance where they may not have necessarily done so before. The XP practice of refactoring entails a user simplifying code whenever an opportunity presents itself [2], but ClearCase's requirement of having developers update their own views causes this practice to be notable obstructed.

3.3 Continuous Integration

XP's continuous Integration practice is executed slightly differently when it is being accomplished with ClearCase. XP dictates that the team members upload their code to the software repository and test it every few hours. [2] One of ClearCase's functions is integrating all of the new system components together; thus actual integration is not performed. As exhibited in Figure 3, continuous integration is implemented with ClearCase by having individual members of the team log on to the integration machine and use ClearCase to upload that machine with the changes that have recently performed every few hours. Although these modifications have passed the component's individual tests, it is at this time that all of the system's tests are run against the changes, which, as seen in the figure, occurs after the build machine receives the source.

The developer can also utilize this opportunity to run the actual application to determine if the program is functioning as expected, despite the new amendments. Although this is performed by the customer in the figure, developers often assume the role of developer.

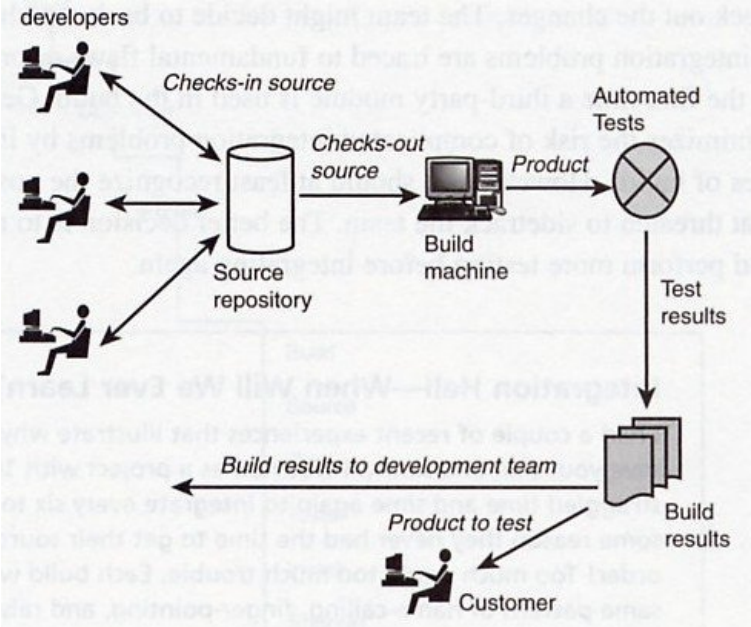

Figure 3: Continuous Integration and ClearCase. Taken from [1]

3.3.1 Benefits

The most evident advantage concerning the use of ClearCase and the continuous integration practice is the fact that ClearCase automates the process by integrating all software that is checked in. Developers need only to update their local project views in order to obtain the newest version of the application. If a development team chooses to have a machine dedicated to testing, then that machine can simply obtain the latest version via ClearCase. Integration is constantly performed, which fits ideally with what this particular XP practice strives for.

Another benefit of using ClearCase in regards to continuous integration is ClearCase has an option of baselining a project. Baselining

can be defined as creating an exact duplicate of an entire project and then storing that project and information regarding that project in a separate location inside of the view. Rather than actually copying that project and storing it in memory somewhere else, baselining stores that information in a view and makes it accessible to the appropriate users. Project managers then set certain dates that the baselining of an application will occur, and that baseline is considered a stage in a project. This process is relatively seamless, and it assists in the XP practice of continuous integration because ClearCase enables baselining to occur with no consequences for the developers. Unlike other SCM tools, ClearCase allows developers to integrate their code regardless of when a baseline occurs.

3.3.2 Shortcomings

The main problem that arises when using ClearCase concurrently with the continuous integration practice is that, in instances where the code being developed is not for an application, the concept of having an integration machine becomes superfluous and, as a result, many development teams choose not to have a machine dedicated to integration. Although this may not initially appear as a negative, it influences teams more heavily than they sometimes realize. When project teams no longer go through the exercise of uploading and testing their code on a third-party machine, whole system and application unit testing often becomes neglected.

3.4 Collective Ownership

ClearCase affects the XP practice of collective ownership by allowing all members of a development team to check out and modify any files situated in a view. Project members are able to examine any

component of the project by updating their local view and scrutinize the files by opening them.

3.4.1 Benefits

The facilitation of continuous integration via ClearCase is accomplished quite clearly. The very nature of the product supports multiple users working on the same product. Developers can learn about the entire system through viewing files that are situated on their views. Once an adequate understanding of the project components is obtained, they then take responsibility for the whole view, as dictated by the practice. [2] All developers are encouraged to check out and refactor any files situated in the view as long as the code passes the tests. Thus, collective ownership is achieved.

3.4.2 Shortcomings

The one failing that ClearCase has regarding collective ownership is that only a ClearCase administrator can disallow a file to be edited. In XP, it is usually not recommended to edit a system's tests after they have been completed and implemented. The instigation of modifying one or more tests should occur only when there is a serious amendment in the functionality of a program. It would be prudent to lock these tests files in such a way that only the owners of the test would be able to modify them. For security reasons, the majority of projects allow only ClearCase administrators to modify file permissions in the aforementioned manner. Since most project teams do not have a ClearCase administrator present, at least in big companies, any member of the team can modify the tests without informing the other developers. The XP practice dictates that by changing the tests, you are changing the functionality. [1] Since a modification of functionality is not a trivial matter, this situation can generate some detrimental results to an application.

4. Conclusions

It is evident that the benefits of utilizing ClearCase in an XP implementation notably outweigh the weaknesses; thus, ClearCase is an effective and practical XP tool. Once ClearCase has been correctly integrated into a project using the XP methodology, each of the 4 practices that it affects is facilitated in such a way that the developers are barely aware that they are performing them. This ensures that these 4 practices are accomplished seamlessly and effectively. Although the advantages of using ClearCase in an XP project are considerable, there are also a number of failings that must be acknowledged.

The shortcomings that arise when using ClearCase in conjunction with a project that is using the XP software development methodology are sometimes disregarded due to the blatant advantages that exist. To ignore the inadequacies in ClearCase regarding XP would be a mistake. Significant deficiencies that develop are the need to obtain the latest versions of the tests whenever a component is being added or modified, the tendency of project teams to neglect the integration process due to ClearCase's functionality, the requirement of updating a view each time a developer performs refactoring on any component in the project, and the entire team's ability to use ClearCase to change the tests and program functionality when it may not be appropriate or necessary. These issues must be addressed by CMI in order to ensure that the synergy between ClearCase and XP is maintained.

5. Recommendations

The shortcomings associated with using ClearCase alongside the XP development methodology must be addressed. If they are overlooked, then problems may cultivate to such an extent that the XP process will be seriously hindered.

The first issue that must be dealt with is the requirement of obtaining the latest version of a component's tests whenever a developer chooses to append or modify a file. The best solution to this would be to have only specified members of the CMI's development teams modify the tests at precise times. This will allow developers to be able to retrieve the system tests at a defined interval, say every two weeks, thus eliminating the need for continually verifying that one's tests are consistent with the latest version of the application.

Another shortcoming that is common among big project teams, such as the ones at CMI, is the failure to accomplish adequate and continually integration. The way this can be rectified is by having a machine that is dedicated to integration and testing, as defined in the XP practice. This machine should be mandatory for all development teams as a means of ensuring that sufficient system and user testing is performed.

Although there is no definitive way of dealing with the quantity of ClearCase view updates required when developers refactor components of a system, the effect can be somewhat negated by having developers only perform refactoring when absolutely necessary or as a phase of project. Even though it is prudent to perform refactoring on a continual basis, the amount of time saved performing ClearCase view updates would make for an equitable trade.

The last concern that should be dealt with is finding a way to restrict a developer's ability to modify a component's test when it is not essential or suitable. Companies that have only a few ClearCase administrators, like CMI, should give at least one member of the team, probably the head developer, administrator permissions for the purposes of allowing some restrictions to be placed on when and which files are edited.

Any other conflicts that arise between ClearCase and XP should be dealt with swiftly and seriously by CMI in order to ensure that the interaction between ClearCase and XP is as successful as possible.

References

[1] S. Baird, *Sams Teach Yourself Extreme Programming in 24 Hours*, Sams Publishing, 2003.

[2] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.

[3] Rational Software Corporation, "Developing Software", 2003; http://publibfp.boulder.ibm.com/epubs/pdf/12653230.pdf (current April 29, 2004)

[4] B.A. White, *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction*, Addison-Wesley, 2000.

Acknowledgements

At this time, I would like to acknowledge Mirek Zyskowski of CIBC Mortgages Inc. for increasing my comprehension of Rational ClearCase and the way it can be used to efficiently create software. Furthermore, I'd like to thank him for teaching me the way software development is accomplished in larger companies.

I would also like to thank Frankie Stephan for the assistance she provided via proofreading this report.

Appendix A: The 12 Practices of XP

The main idea behind the XP software development methodology is taking pre-existing development activities and performing them as intensely as possible. This is achieved via the 12 XP practices that are not revolutionary individually; rather, it is the conjunction of all these practices that makes XP unique. Table 3 provides a summarization of all these practices as defined by XP's creator, Kent Beck. These practices are not new, but they have been modified in such a way that they fit with the XP methodology. [2] An example of an XP variation on a practice is instead of testing after code is completed, testing is executed all the time.

Practice	Description
The Planning Game	Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan.
Small releases	Put a simple system into production quickly, then release new versions on a very short cycle.
Metaphor	Guide all development with a simple shared story of how the whole system works.
Simple design	The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.
Testing	Programmers continually write unit tests, which must run flawlessly for development to continue.
Refactoring	Programmers restructure the system without changing its behaviour to remove duplication, improve communication, simplify, or add flexibility.
Pair programming	All production code is written with two programmers at one machine.
Collective ownership	Anyone can change any code anywhere in the system at a given time.
Continuous integration	Integrate and build the system many times a day, every time a task is completed.
40-hour week	Work no more than 40 hours a week as a rule. Never work overtime a second week in a row.
On-site customer	Include a real, live user on the team, available full-time to answer questions.
Coding standards	Programmers write all code in accordance with rules emphasizing communication through the code.

Table 3: Summarization of the 12 XP Practices. Taken from [2].

XP emphasises performing 12 practices that, as seen in Figure 4, interact and depend on each. If one practice is performed well than other practices will be affected positively, and if one practice is neglected than the rest of the practices are affected negatively. Creating and maintaining this synergy is essential to the success of any XP project.

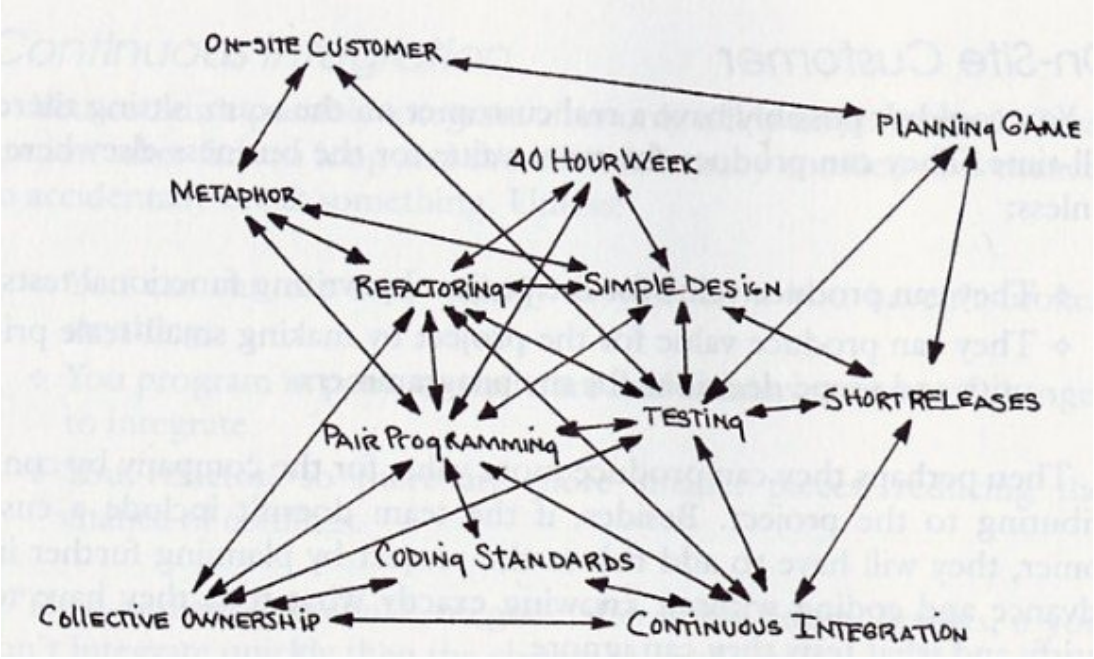

Figure 4: Synergy between the XP Practices. Taken from [2]