|  |
| --- |
| **Team Effort**  **Soccer Team**  **Management System**  Milestone 3 |
|  |
| **Luke Brisebois - lub224**  **Simon Fanner - saf725**  **Steven Hancock - smh875**  **Adam Mravnik - ajm207**  **Amin Shakev - ams162**  **Patrick Weckworth - paw818**  **Tom Wetzel - thw740**  **Drake Zarowny - djz587** |
|  |

|  |
| --- |
|  |

Table of Contents

1. Introduction 4

1.1 System Description 4

1.2 Current System Status 4

2. Design Artifacts 4

2.2 Implementation of Best Practices 4

Binary Mini Milestones and Estimations (See Appendix D) 4

Risk-Driven Incremental Delivery Overview 5

2.3 Implementation of Management Tools 6

Team City (See Appendix E) 6

Clone Detection Using NiCad and VisCad 7

Emma (See Appendix H) 8

JMock (See Appendix J) 9

Log4J 10

Maven (Appendix I) 10

3. Milestone (Revised) 11

3.1 Previous Milestones 11

3.2 Current Milestone 13

3.3 Future Milestones 14

4. Risk Evaluation 14

5. Team Personnel 14

6. Milestone Activity Report 14

6.1 Work Completed 14

6.2 Git Log 14

6.3 Wiki Log 15

7. Conclusion 15

Appendix A 16

Risk Report 16

Risk Re-evaluation 18

Appendix B 19

Appendix C 23

Appendix D 25

Appendix E 27

Appendix F 27

Appendix G 27

Appendix H 27

Appendix J 27

Appendix J 28

# 1. Introduction

## 1.1 System Description

The project chosen for the class is a Soccer Team Management System, developed by TeamLeader in CMPT 370 in 2011. The software is a web-based system which relies on a database to store and access all types of information. A variety of programming languages are utilized in the project including Java, JavaScript, MySQL, CSS, and HTML. The project was developed in NetBeans IDE and will be done so for this class as well.

## 1.2 Current System Status

The system has not changed since the last milestone.

# 2. Design Artifacts

This section describes the changes made to the project compared to the previous milestone.

## 2.2 Implementation of Best Practices

### Binary Mini Milestones and Estimations (See Appendix D)

**Binary Mini Milestones**

The use of Binary Mini Milestones and estimation practices are extremely important for . These two best practices will mainly focus on the administration and management sides of the project and the group. Both of these tools are extremely helpful in keeping projects on track. They allow for ways to monitor and estimate time requirements for tasks and the completion of these tasks. Changing to Binary Mini Milestones will change the group dynamic and should help improve communication and production, in addition to changing the time management and risk analysis. Using the key best practice of estimation will help estimate time and risk requirements for the rest of the project this term.

To help better manage our team, we have implemented Binary Mini Milestones for this Milestone 3. These Binary Mini Milestones will help better define each group members tasks and specific work and will also provide an easy and accurate way in which to monitor our progress for this milestone. The Mini Milestones are as follows are listed in **Appendix D**.

**Estimations**

For Milestone 3, Range Estimates were chosen for the numerous benefits they will provide. They explicitly differentiate best versus most likely versus worst case scenarios for each task. They will make it less likely that individual group members mistake worst case as a most likely case. Range estimates provide our client with important information regarding the uncertainty of estimates and will decrease the risk that an estimate is considered to be a commitment. For this milestone, all estimation practices will focus specifically on this milestone and each of the previous Binary Mini Milestones that are outlined above.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Hours to Complete | | | | |
| Name | Item | Best Case (25%) | Most Likely Case | Worst Case  (25%) | Expected Case (50%) |
| Drake | Maven | 2 | 2.5 | 5 | 3.16 |
| Adam | JMock/Log4J | 3 | 4 | 7 | 4.67 |
| Simon | DB/CI/ST | 3 | 3.5 | 4 | 3.5 |
| Luke | Mini MS/Est | 3 | 3.5 | 4 | 3.5 |
| Patrick | EMMA | 4 | 4.5 | 5 | 4.5 |
| Steven | VisCad/Nicad | 4 | 5 | 7 | 5.33 |
| Tom | Peer Review | 2 | 3 | 5 | 3.33 |

### Risk-Driven Incremental Delivery Overview

The risk-driven incremental delivery practice draws heavily from the spiral model method of software development.  The separate parts present in each iteration of the spiral are:

* Planning: estimation, scheduling, and risk analysis
* Modeling: analysis, design
* Construction: code, test
* Deployment: delivery, feedback
* Communication

It is a systematic approach combined with the iterative process of prototyping.  Due to the way requirements are refined and prototypes hone in on the final product, there is a decrease in risk through each cycle of the spiral process.  The importance of planning after each iteration is crucial, so that the final product will represent what the client's goals were from the outset.  The cyclic nature of the process was created in response to the pitfalls of the waterfall model.  The incremental delivery method also allows for the results of the work to be seen much faster, as there are regular prototypes released.  It is important to note, that each prototype must be completely useable, and must serve some use to the users.  Functionality should be increased each time, preferably with the most important core functionality implemented first.

The risk analysis should serve as a way to determine which functional requirements are plausible and which are not.  It should also serve to determine at what point in the project life cycle a particular feature would be constructed.  By managing the risk levels, the chances of project failure will be minimized.  Within each of the phases, there are also several iterations that can take place, depending on the time schedule of the project.

Originally, when the group first came up with our goals and project plan, Team Effort had decided to implement a large amount of new features, and generally overhaul the whole Soccer Team Management System.  It was quickly discovered, by the second milestone, that this was not the Professor's intention for us in this class, so the amount of coding to be done was scaled back immensely, in favour of learning about tools and best practices.  That being said, the code has mostly been neglected with the exception of extensive testing to this point.  By attempting to employ a small amount of Risk-driven Incremental Delivery to the remaining milestones, the group can gain knowledge and experience using the best practice, as well as add a bit of functionality to the software.

For this current milestone, the planning stage was completed.  It was decided that the features or functionality to be added should be relatively low risk and small so that they are not overly time consuming, but still significant enough to employ the best practice effectively.  A few ideas from the original project plan were looked at, as well as a few other ideas.  Below is a possible guideline for how the practice will be used throughout the rest of the term:

**Revamp of user login/password system to increase security**

* This should be a relatively small-scale job as it is primarily going in and modifying already existing code
* Risk level: Low
* Time estimation: 0.5-1 weeks to complete.  Only 1 programmer needed.

**Addition of league selection for the system**

* This is a slightly more coding intensive job, as it would require modifying the database structure and addition of new functionality
* Risk level: Medium
* Time estimation: 1-2 weeks to complete if assigned to a pair of programmers.

For whatever functionality is chosen, the group will attempt to document the progress based on the 5 stage process outlined above.  The results can then be evaluated to determine if the best practice was followed effectively.

## 2.3 Implementation of Management Tools

### Team City (See Appendix E)

TeamCity provides a platform for developers to achieve continuous integration and keep track of test success, build failures, or dependency issues.  It allows for multiple integrations per day, and automatically performs each build and test to allow for quick detection of errors.  This in turn means that developers are notified immediately when something has been broken, and the support is there to roll back to a previous version of the build.  The overall user interface is fairly well polished, however it would be nice to see an even more "dumbed-down" version for those that are completely new to Continuous Integration software products.  There is support for a wide variety of operating systems and many languages and a plethora of features.  From what other resources, if a build fails, TeamCity will provide you with a link to open the broken code segment directly in the chosen IDE.  There are also detailed reports for each build, including the number of tests passed, failed, or ignored.

As for using our Soccer Management System project with TeamCity, the lack of a plugin for NetBeans greatly increases the difficulty of doing so. It must be possible to implement it somehow using a variety of other software suites, however, the difficulty greatly increases when trying to create various build steps using runners that the team is unfamiliar with.  The team is considering trying to implement a smaller scale sample project, possibly with the available Eclipse plug-in for a future milestone, or at least looking more into how TeamCity could be incorporated to our existing project.  Also, discussion for integration with Maven for using Maven based build configurations is a possibility.

Overall, TeamCity seems like a very comprehensive and complex tool.  Unfortunately, it seems understanding it thoroughly will be a task that is outside the available time and scope of this course.

### Clone Detection Using NiCad and VisCad

Clone detection is extremely important in eliminating software maintenance costs. Duplicate code is the number one *bad smell* for software code. Duplicate code can be injected into code for numerous reasons. Clones can be cause by developers using copy and paste, merging similar systems, and accidentally or unknowingly implementing the same logic by different programmers. Code clones can be dangerous for many reasons:

* If bugs are found in the duplicated code the bug may only be fixed in one location
* Source code can become bloated with repeated lines of code
* New features may have to be introduced into multiple files
* It becomes difficult to understand the difference between clone fragments

Because clones have the potential to be so dangerous, and thus cause software maintenance costs to skyrocket, it is important to detect and minimize clones.

**NiCad**

NiCad is an excellent tool for clone detection. NiCad is a program, based off of TXL that compares fragments of clones against one another and determines, based on a specified threshold, if the fragments are similar enough to be considered duplicates of one another. NiCad has different options that affect what is reported. The *config/default.cfg* file contains most of the options. Here is a list of some of the more important ones:

* *Threshold*- The threshold of clones to report. The smaller the threshold the more clones reported. It determines the % of line difference between the two clone pairs.
* *Minsize \_and \_maxsize* - The min/max size of clones that are detected (in terms of line numbers).
* *Report* - Make XML and HTML source reports (yes or no)

NiCad's XML reports can be used by many any XML parsing program to display the results in a coherent manner. However, NiCad's most readable report is its HTML reports that are viewable by any browser. The HTML report displays things like total functions found, clone pairs found, lines of code compared, CPU time to produce the report, and the number of classes analyzed. For Team Effort's soccer system, the following results were given Total Functions: 214, Clone pairs found: 42, LCS compares: 769, CPU time: 0 min 0.100 sec, Number of classes: 10. The soccer system is a complex system that uses multiple languages such as CSS, HTML, JavaScript, JSP, and Java. Team Effort is only analyzing the code clones for the Java source code to reduce complexity. The HTML report produced by NiCad also produces the clone class number, the number of fragments that match the clone class, the size (in terms of number of lines), and the degree of similarity between the fragments, for every clone that is detected. With NiCad, Team Effort is able to determine the current clones in the software system. With the code clone information Team Effort can take measures to eliminate clones and minimize future software maintenance costs. More information about NiCad can be found in **Appendix F.**

**VisCad**

Detection of code clones is the first step in eliminating clones. However, detecting them is not enough. Clones should be eliminated one by one, piece by piece. To determine which clones should be eliminated first, Team Effort used VisCad to visualize the report created by NiCad. VisCad is a visualization tool that allows users to analyze large volumes of raw cloning data in an interactive fashion. Users can analyze and identify distinctive code clones through a set of visualization techniques, metrics and data filtering operations. The use of VisCad is relatively simple, however, the report produced by NiCad3 is not in the proper format for VisCad to read, so some intelligent editing had to be done to reformat the report. Once VisCad was able to interpret the report, Team Effort was able to analyze the current state of the soccer system.

For a more in-depth analysis of clone detection see **Appendix G**.

### Emma (See Appendix H)

EMMA is a tool which seems to be very good at identify the actual coverage of any sort of tests done on a piece of software. After working out some issues with it and Netbeans and JUnist, and getting them all working together, it was fairly easy to install and implement. In Netbeans it appears to work excellently with Java applications which have implemented JUnit tests, and could potentially generate some genuinely useful data. It was simple as a activating it, running your tests, and viewing the coverage report generated. The report gives very detailed information about which packages, classes, and lines are actually hit during test execution, and also some estimation of whether or not coverage was sufficient for testing purposes, although by the looks of a generated report it appears to just check for 50% coverage.

Unfortunately this tool was not overly useful for this particular project. First of all since it is a web application, and not a Java application, it is not even possible to use EMMA on the project with out ripping it apart and mashing together a Java application with what is left. For this project that meant getting rid of all web pages such as JSP's, Javascript libraries, style sheets, and Servlets, despite the fact that Servlets are just Java code. Fortunately this leaves plain Java classes and the JUnit tests that apply to them. This method means that whenever a code coverage report is desired on current code, all of the appropriate files have to be copied into a Java project, and it also means very little of the project is actually tested since it is a web project with more than half of its content being inappropriate for this tool.

Using this method and running it on the current project (Mar. 10, 2012) generated the following results: package coverage: 60%, class coverage: 55%, line coverage: 32%. The plugin determined from these results that the project is covered. The coverage report is basically a rating of the unit test, and as it only knows about plain Java classes does not really show the whole picture. Even if that was the whole project though, it seems as if just over half of the classes and packages does not really mean that the project is covered well enough yet.

One note about this tool is that it does not have to be used as a plugin for Netbeans. It can be used, and was briefly used, from the command line on JAR files. In this manner EMMA can be used to show the code coverage of regular usage rather than the coverage of unit testing. This does seem to work fairly well, however in the very brief time spent with it in this way, it did seem to have trouble dealing with multi-threaded applications. This project was never run in this way since it can't actually run outside of unit testing without the web pieces.

### JMock (See Appendix J)

JMock allows developers to create “Mock Objects” that simulate real objects in Java. The main benefit of this is that it can allow classes to simulate other classes and can help to avoid dependency issues when testing. A further example would be if Class A depends on Class B but Class A is what needs to be tested, then “Mock Objects” can mimic the behavior of Class B without needing to use Class B and thus resolving dependency issues.

JMock is a tool that allows you to apply mock objects in JUnit for test-driven development. It is particularly useful for testing things with external components and is great at speeding up Unit Tests by mocking out heavy dependencies.

JMock was used in this project mock out connections required to run some of the more important servlets. Servlets are normally a pain to unit test but by using JMock as seen in the code paste above we were able to test that the structure of servlets stay consistent.

An intereting point to note would be that JMocking servlets is a little bit counter intuitive. It is very hard to write tests before hand for a servlet in this way because one would have to know absolutely everything they were going to write and not just the logic behind it. However, in the end it proved to be an essential tool in testing the servlets in this project by checking for consistent values and structure of the code while not needing to send and receive actual HTTP requests.

In summary, JMock is a great tool. It speeds up test-driven development and allows certain aspects of code to be tested that is usually more difficult with traditional methods.

### Log4J

Log4J is a tool that enforces the best practice of Rigorous Risk-Driven Testing. It is a tool that will allow us to log all errors, print statements, warnings, etc, in our project and sort them by priority and importance. Log4j, or commons logging, is a jar library that is supposed to improve the Java JDK Logging API and be used as an alternative. Log4J proved itself problematic in glassfish with our project. Research has shown that if the project was migrated to apache tomcat that this problem would be resolved and as such this was attempted.

Apache Tomcat was a challenge to get to work with Netbeans 7 and time was spent researching why that may be. It was attempted to get log4J to work under glassfish by following some online guides that involve adding log4j and log4j.properties into the Glassfish\_Home/lib/ directory. This did not work and no solution was found that was not dependent on the one system it is running on. Log4j fails to execute any calls when it is run because of this issue.

The log4j library was been added to the project and committed to Github but subsequently removed. Log4j's issues outweighed it's benefits for the case of this project and although multi-layers of logging is useful, when working with servlets it is also an unnecessary hassle that will not provide many gains.

The issues described above have resulted in a failed implementation of log4j but a lot of knowledge was gained on how it works and why it was not ideal for this project.

### Maven (Appendix I)

Getting started was easy because NetBeans is packaged with a Maven plugin with is great, but it easier to manually add all the libraries that were need from the command line. This can be tricky especially on Windows, as GlassFish is an installed program.  There is a plugin that apparently works with the GlassFish install and Maven but the team could not get it working, so a new fresh install of GlassFish had to be completed but as a fully embedded jar which could be added to the Maven repository.

Maven has a lot of build options, however, because of the team’s limited knowledge of Servlets and JSP the team has yet to modify the build options. Maven does build and test (JUnit) Team Effort’s project nicely.

Maven is mostly meant for very large projects with many build options and a comprehensive test suite, so there isn’t a lot to gain in the soccer system project. Looking after dependencies is always a pain so it’s great that Maven can do that for the project,  but even with GlassFish Team Effort’s soccer system only has four dependencies in total for our project.

# 3. Milestone (Revised)

There will be five major Milestone deliverables and five presentations over the course of the term. Each Milestone will include some new functionality as well as its related testing. Below is a Mini-Milestone list of the proposed objectives. Future milestones objectives have been altered to better reflect the need for management and tool implementation as opposed to design implementation. Previous milestones objectives have not been altered, however conclusions have been added, all other milestones are being refined in this document and in future milestones, removed object are in red, added objectives are in green.

## 3.1 Previous Milestones

**Milestone 1**

*Objectives*

* Resolve technical issues with database and versioning system
* Begin detecting existing bugs, defects, and limitations of program
* Develop plan for subsequent milestones

*Conclusions*

* Had previous database restored and migrated to Team Effort's database.
  + Made backup copies, and will continue making backup copies to ensure problem doesn't arise again.
* An SVN repository was set up, but accessibility was difficult. For Milestone 1 we used drop box until our SVN accessibility issues were fixed.
  + SVN as a version control system has been dropped in the current Milestone (Milestone 2) in favor of GitHub.
* Bug detections and determining application limitations is ongoing.
* Revising plans for upcoming milestones is ongoing.

*Time Sheet*

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Time Spent | I worked on… |
| Luke Brisebois | Jan 24 | 1 Hr | Reviewing Soccer Project  -running the system  -looking at code  -familiarizing myself |
| Feb 2 | 2 Hr | Adding to report |
| Simon Fanner | Jan 31 | 1 Hr | Report/Presentation Outline |
| Feb 1 | 2 Hr | Report documentation/Tech issues |
| Feb 2 | 2 Hr | Database dump/Report completion  - handed in Milestone 1 to moodle  - may need/have time to revise it |
| Steven Hancock | Feb 1 | 1.5 Hr | TXL Pretty Printing |
| Feb 1 | 1.5 Hr | Risk Analysis |
| Feb 3/4 | 0.5 Hr | Just a little bit more on TXL, and a small write-up |
| Adam Mravnik | Feb 1/2 | 2 Hr | Tried to resolved ssl and access issues with svn  by communicating with CS Trac and modifying my own SSL settings. |
| Feb 2 | 0.75 Hr | - Restructured the project files in Dropbox.  - Removed all old svn references from when it was a 370 project.  - Renamed the project and various files to make their purpose updated  and more clear.  - Imported the project into the SVN repo using Tortoise SVN  - Current SVN Status: Works in windows, and on tuxworld (possibly all linux distros?),  does not seem to work in OSX |
| Amin Shaker | Jan 18 | 0.5 Hr | Presentation and reviewing the database |
| Feb 1 | 1.5 Hr | Working on presentation and milestone goals |
| Patrick Weckworth | Feb 1 | 3.5 Hr | Hours writing report |
| Tom Wetzel | Jan 18 | 0.25 Hr | Setting up wiki pages |
| Feb 1 | 2 Hr | Working on presentation and milestone goals |
| Feb 2 | 0.5 Hr | Helping connect to SVN and the database, presentation meeting |
| Drake Zarowny | Feb 1 | 2 Hr | Working on presentation and milestone goals |

**Milestone 2**

*Objectives*

* Tool Bar/Menu for navigation
* Implementation of unit testing using Junit
* Develop a more extensive bug tracking tool
* Use TXL to format code Java code into single style
* Use of refactoring in Netbeans in order to better structure code
* Restructure and refine database (in code and in database)
  + Revise password authentication (more secure, terminate sessions)

*Conclusion*

* Team Effort was able to complete all of the objectives for Milestone 2.
* Team communication difficulties made the Milestone more difficult than it should have been. The lack of communication, and when there was communication, miscommunication created a situation where Team Effort had difficulty completing the Milestone on time and efficiently.
* Following milestones should put more focus on the management tools and following best practises.

*Time Sheet*

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Time Spent | I worked on… |
| Luke Brisebois | Feb 18 | 2 Hr | Reviewing Milestone Document |
| Feb 25 | 2 Hr | Preparing for presentation |
| Simon Fanner | Feb 15 | 1 Hr | Setting up Git, re-factoring project name |
| Feb 17 | 0.5 Hr | Bug Documenting on Wiki |
| Steven Hancock | Feb 9-13 | 1 Hr | Getting Git setup and importing the project |
| Feb 13 | 0.5 Hr | Putting together a Doc for M2 |
| Feb 13 | 0.5 Hr | MySQL dump for backing up database |
| Feb 13/14 | 4 Hr | Writing milestone document |
| Feb 15 | 1.5 Hr | Creating menu navigation |
| Feb 15-18 | 5 Hr | Finishing milestone document and prepping for handin |
| Feb 25 | 2 Hr | Creating Presentation |
| Adam Mravnik | Feb 14 | 1 Hr | Writing test cases and setting up Git |
| Feb 17 | 2.5 Hr | Wrote Junit tests for schedule and stats |
| Feb 18 | 1 Hr | Wrote userBean tests and did some git research |
| Amin Shaker | N/A | N/A | Nothing Reported |
| Patrick Weckworth | Feb 16 | 2 Hr | Struggled with and finally set up git |
| Feb 16 | 2 Hr | Wrote a test class with some test |
| Feb 16 | 0.5 Hr | Wrote a bug report |
| Tom Wetzel | N/A | N/A | Nothing Reported |
| Drake Zarowny | Feb 20 | 0.5 Hr | Getting Github version of project setup in netbeans |

## 3.2 Current Milestone

**Milestone 3**

*Objectives*

* Implement NiCad to determine software clone's in the project and use VisCad to visualize clones
* Implement logging feature
* Implementation of management tools like TeamCity, jMock, Emma, Log4J, Apache Maven, and Jira
* Implementing best practices, peer review, risk driven incremental delivery, and binary mini milestones and estimation practices.

## 3.3 Future Milestones

**Milestone 4**

*Objectives*

* Make system more user-friendly
* Use VisCad to help visualize, manage, and order clones detected by NiCad
* Implementation of additional tools like Checkstyle, GCad, etc

**Milestone 5**

*Objectives*

* Different view styles on profiles or stats (sort by teams etc)
* Player email notification
* More robust messaging system
* Implementation of additional tools like Checkstyle, GCad, etc

# 4. Risk Evaluation

**Appendix A** contains the risk report. No changes have been made to the risk report from the previous milestone.

# 5. Team Personnel

For the third milestone the team’s roles have been redefined in addition to the leader, mostly a figure head, also changing. The leader in this milestone will be Steven Hancock who will be the main contact person for the group as well as in charge of handing in the milestone.

# 6. Milestone Activity Report

This activity report outlines what has been accomplished by the design team over the duration of the milestone.

## 6.1 Work Completed

As mention previously, new tools and new best practises have been implemented in this milestone. However no code development has occurred.

## 6.2 Git Log

Please see **Appendix B**.

## 6.3 Wiki Log

Please see **Appendix C**.

# 7. Conclusion

Team Effort has put the major efforts into implementing new tools and new best practises. The new tools allow for better management of the project, in building, deployment, testing, and source code management. The new best practises that are being implemented and follow will allow for more reviewing, better time management, and lower risk.

# Appendix A

## Risk Report

*Software Requirement Risks*

* Change of requirements
  + The requirements of the software have drastically changed. Team Effort has removed many of the planned additions that we had laid out in the first milestone and shifted focus towards implementing new tools to help manage the software.
* Poor definition of requirements
  + The requirements from the previous milestone were some-what vague and had to be further revised for this milestone such as “What is a navigation bar? What should it include?” and “What is restructuring the database?”
* Impossible requirements
  + All requirements were possible

*Software Risks*

* Project & Milestone completion dates being unrealistic
  + Realistic given an isolated class, however with many assignments due and Mid-terms around the same time as Milestone 2 is due makes allocating time to this assignment difficult.
* Hardware (server issues)
  + Not a problem
* Lack of Testing
  + Not a problem
* Personal differences in design/coding techniques
  + Since there is less focus on design and more focus on management this risk is easily avoidable, especially with the help from TXL to format the code in one singular way.
* Lack of knowledge could make some features unobtainable
  + Not a problem
* Human Errors
  + None have occurred

*Software Scheduling Risks*

* Over-estimate time requirements
  + Not a problem
* Under-estimate time requirements
  + The group has severely under-estimated the time it requires to coordinate activities and to split up the work. Weekly meetings have not been enough, especially when its not always possible for everyone to make the meetings.
* Not managing time affectively
  + Because there is an under-estimation of time requirements the group needs to work better at maximizing the time that they do have together. Things like being better prepared for every meeting and participating on the project’s Wiki will help.
* Requirements changing and not being able to adequately allocate time
  + Not a problem
* Lack of skill could require additional learning to implement goals
  + Not a problem
* Tool failure, like SVN, or difficulties with NetBeans or Java Server
  + This has been the team’s single biggest hurdle. SVN has given Team Effort headache’s and the issue took weeks to resolve, and the resolution is still a little ‘iffy’. In the migration to GitHub there have also been difficulties as learning on the fly how to use a new version control system is not ideal.
* Lack of knowledge of tools
  + The project management tools that Team Effort would like to are all to new to the group. So time will have to be allocated for individual members (and the group as a whole) to learn the new tools in order to implement them into the project.

*Software Quality Risks*

* Improper or lack of design documentation
  + Not a problem
* Unrealistic scheduling leading to lack of testing and deploying bug filled application
  + Not a problem
* Lack of knowledge leading to unforeseen bugs, errors, or unexpected results
  + The group has limited knowledge of new tools. The new tools that are being introduced to the project bring the possibility of introducing new errors, bugs, and unexpected results.
* Application’s user interface not easy to use
  + This is a minor issue, however a navigation bar, would help to make the interface easier to use. For this reason, a navigation bar will be implemented in this milestone.

*Team Risks*

* Lack of communication
  + Communication is becoming a problem. Participation in the Wiki is not as active as it could be, classes and meetings are not being attended by all, and outside of class, group meetings, and the Wiki sees little to no communication.
* Scheduling conflicts
  + It has been very difficult for Team Effort to arrange any meeting times outside of the already scheduled weekly meeting.
* Lack of responsibility (ownership)
  + Those participating are taking ownership.

*Software Business Risks*

* No one wants the application
  + This is not applicable as we are not trying to sell/distribute the application.
* Budget failure (time or financial)
  + It remains unclear if the budgeted time we have given will create a project failure.
* Distribution failure
  + Distribution will most likely not occur in the form of Developer to User.

## Risk Re-evaluation

In light of our possible risks, the team will take a number of actions to ensure maximum risk avoidance. The requirements will be outlined as complete as possible in the first Milestone, however, Team Effort will remain flexible in case new requirements, or requirement definitions are changed. Any requirements that are deemed to be too expensive or impossible to meet will be abandoned immediately and no more time will be allocated to those requirements. Team effort will always conservatively budget time to ensure time-cost over-runs are minimal. The team will also ensure sufficient time is allocated for testing. Through code reviews and TXL ‘pretty printing’ the software’s code, although written by multiple programmers, will all be uniform.

# Appendix B

commit eb167fe6dbdcd01c3098c1934252bb1a8b04a13d

Author: Adam Mravnik <a.mravnik@usask.ca>

Date: Sun Mar 11 20:12:03 2012 -0600

Wrote jmock tests for the more important servlet functions

commit ff9260956481c1c018fb7ecd22c3719339f23c19

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Tue Mar 6 12:35:04 2012 -0600

Added Clone files

commit 73d536f0b5509ac50b28fd4c6cb1db8024b64782

Author: Simon Fanner <saf725@mail.usask.ca>

Date: Wed Feb 22 15:24:06 2012 -0600

Testing Github yet again 1

commit 6cd1b08f08706240a119a39ebf346588b7bd006d

Author: Simon Fanner <saf725@mail.usask.ca>

Date: Wed Feb 22 14:58:57 2012 -0600

Testing Github yet again

commit 71bd967cf9409b3e876f937ddc87ca666774526a

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Sat Feb 18 20:12:51 2012 -0600

Finished report

commit ae28b42630266daa71c6f171b0ddfb7b2e64854c

Author: Adam Mravnik <a.mravnik@usask.ca>

Date: Sat Feb 18 11:42:24 2012 -0600

Wrote the userBean tests

commit 587d4b85d84a91937e0afd899fd75b91f015e50b

Author: Adam Mravnik <a.mravnik@usask.ca>

Date: Fri Feb 17 21:59:02 2012 -0600

Added stats tests and moved it in with the edit schedule tests because it is necessary to know what game I am working on

commit df1ff82e6dfe8b558306d561246c396c90acb35a

Author: Adam Mravnik <a.mravnik@usask.ca>

Date: Fri Feb 17 21:39:22 2012 -0600

Wrote editSchedule Tests

commit 8c537180301e90d4d3096d557f8750573f6f29d9

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Fri Feb 17 09:34:22 2012 -0600

Added menu

commit a0449dc2a4a432bd762342326094d4bf0b20f07e

Author: paw818 <paw818@peon32.usask.ca>

Date: Thu Feb 16 19:59:47 2012 -0600

Created a test class for editRoster.java and wrote some tests for it.

commit 04472a9d2d45e28a2a192d103204a462ca8636f2

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Thu Feb 16 18:58:01 2012 -0600

Report almost finished

commit 33851d2be9849567fc578b065fc8761a3ed80d69

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Wed Feb 15 19:00:11 2012 -0600

Filled out the Risks part

commit 407314afd397d2ca98e58da08a16f2f16f22f7f1

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Wed Feb 15 17:16:03 2012 -0600

Removing old cmpt370 directory

commit b553f9e13a9601dbd7f7fd208ec999112f23b8b5

Author: simonfanner <simonfanner@10.226.160.136>

Date: Wed Feb 15 16:28:47 2012 -0600

Updated the Project Name to reflect CMPT 371.

(Also testing to make sure I have git set up properly. Hopefully this doesn't blow up)

commit 3a5eb742762e9c8ea7e5d76251e2c4ae7d4898ba

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Tue Feb 14 19:36:10 2012 -0600

Added to Milestone2's report

commit 885b16956b6368e015a36fa891289e095f0b5d87

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Mon Feb 13 22:48:29 2012 -0600

Added Milestone2-Report and filled out some sections and have the skeleton for others

commit edeb2cca86a58a7ed4fc30203f1db43adc5f8b8c

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Mon Feb 13 15:48:31 2012 -0600

Added a mysql dump

commit 6a24f77266506da465521da0940dfd24719ee8b5

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Mon Feb 13 15:29:41 2012 -0600

Cleaning up working directory and adding breif outline for milestone2

commit a238e12e9d02de54d9cf56e1a38f1f4d42409cd9

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Mon Feb 13 12:20:20 2012 -0600

Renamed M1 Final Report

commit b967aed46a5a93453fea7d936ac46199f512accf

Author: smh875smh <smh875@mail.usask.ca>

Date: Thu Feb 9 14:48:23 2012 -0600

Update README

commit 1a132ea0ca00e12d02e5d4c6acae2bcfe0eef629

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Thu Feb 9 14:39:14 2012 -0600

Updated README

commit 25a86e06d319d293939e8d55775a90c3e96c5071

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Thu Feb 9 12:35:51 2012 -0600

Re-organized image and css files.

commit 7a4ff7bc89bcfd6a86d32d0e70600680749d0bd1

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Thu Feb 9 12:17:25 2012 -0600

Re-organized image and css files.

commit bd848d021367087d2c16fef6b833f999f61ef22a

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Thu Feb 9 12:16:33 2012 -0600

Re-organized image and css files.

commit bfe447e78b5cd246d6894cbbd23c6c9f339bd113

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Thu Feb 9 11:58:10 2012 -0600

Re-organized image and css files.

commit 6ca5833510dc7d237c7c26fe02161c5261d84759

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Thu Feb 9 08:30:43 2012 -0600

Initial import of project

commit 3c849c9156f165b466c2c92e25a55b525cbbcba3

Author: Steven Hancock <smh875@mail.usask.ca>

Date: Wed Jan 25 15:00:33 2012 -0600

First Commit

# Appendix C

Hours sheet - by Luke Brisebois - about 2 hours ago

Hours sheet - by Thomas Wetzel - about 4 hours ago

Hours sheet - by Steven Hancock - about 4 hours ago

VisCad - by Steven Hancock - about 4 hours ago

Roles - by Simon Fanner - about 5 hours ago

Mini Milestone for Milestone 3 - by Simon Fanner - about 5 hours ago

Hours sheet - by Simon Fanner - about 15 hours ago

Risk-driven Incremental Delivery - by Simon Fanner - 15 hours ago

Best Practices - by Simon Fanner - 16 hours ago

JetBRAINS - TeamCity 7 - by Simon Fanner - about 16 hours ago

5.png - by Simon Fanner - 16 hours ago

4.png - by Simon Fanner - 16 hours ago

3.png - by Simon Fanner - 16 hours ago

2.png - by Simon Fanner - 16 hours ago

1.png - by Simon Fanner - 16 hours ago

Hours sheet - by Adam Mravnik - yesterday at 09:59 PM

EMMA Code Coverage - by Patrick Weckworth - yesterday at 05:06 PM

Maven - by Drake Zarowny - yesterday at 04:54 PM

Mini Milestone for Milestone 3 - by Thomas Wetzel - yesterday at 04:41 PM

Hours sheet - by Drake Zarowny - yesterday at 03:56 PM

Log4J Report - by Adam Mravnik - Mar 11

JMock Report - by Adam Mravnik - Mar 11

NiCad - by Steven Hancock - Mar 11

Hours sheet - by Patrick Weckworth - Mar 10

viewCoverage.jpg - by Patrick Weckworth - Mar 10

runTest.jpg - by Patrick Weckworth - Mar 10

activate.jpg - by Patrick Weckworth - Mar 10

Roles - by Amin Shaker - Mar 08

Mini Milestone for Milestone 3 - by Steven Hancock - Mar 08

Roles - by Patrick Weckworth - Mar 07

Meeting Minutes - by Adam Mravnik - Mar 07

Roles - by Luke Brisebois - Mar 07

Roles - by Adam Mravnik - Mar 06

Roles - by Drake Zarowny - Mar 06

HierarchicalDependencyGraph.png - by Steven Hancock - Mar 06

codeClones.png - by Steven Hancock - Mar 06

fileCloneSnippets.png - by Steven Hancock - Mar 06

TreeMap.png - by Steven Hancock - Mar 06

cloneDistribution.png - by Steven Hancock - Mar 06

Tools - by Steven Hancock - Mar 04

Bugzilla How To - by Steven Hancock - Mar 04

Bugs - by Steven Hancock - Mar 03

Roles - by Steven Hancock - Mar 01

Meeting Minutes - by Simon Fanner - Feb 29

Roles - by Thomas Wetzel - Feb 29

Presentations - by Steven Hancock - Feb 25

GitHub and Database Access Info - by Simon Fanner - Feb 22

Password Case Bug - by Adam Mravnik - Feb 18

Table Display Formatting - by Simon Fanner - Feb 17

Password Case Bug - by Simon Fanner - Feb 17

Meeting Minutes - by Steven Hancock - Feb 17

Hours sheet - by Patrick Weckworth - Feb 16

editRoster addUser( false team) - active - created by Patrick Weckworth - Feb 16

tests errors - created by Amin Shaker - Feb 16

GitHub and Database Access Info - commented by Patrick Weckworth - Feb 16

Hours sheet - by Amin Shaker - Feb 16

GitHub and Database Access Info - commented by Steven Hancock - Feb 15

GitHub and Database Access Info - commented by Thomas Wetzel - Feb 15

Roles - by Thomas Wetzel - Feb 15

Bugs - by Steven Hancock - Feb 14

GitHub and Database Access Info - by Steven Hancock - Feb 13

Roles - by Simon Fanner - Feb 09

Roles - by Steven Hancock - Feb 09

Roles - by Amin Shaker - Feb 08

Roles - by Adam Mravnik - Feb 08

Hours sheet - by Luke Brisebois - Feb 08

TXL Pretty Printing - by Steven Hancock - Feb 03

PP-KeepStruct.tar.gz - attached by Steven Hancock - Feb 03

Objectives - by Thomas Wetzel - Feb 02

Bugs - by Adam Mravnik - Feb 02

Hours sheet - by Thomas Wetzel - Feb 02

PrettyPrinting.tgz - attached by Steven Hancock - Feb 01

SVN and Database Access Info - commented by Steven Hancock - Jan 31

Group Admin - created by Adam Mravnik - Jan 31

GitHub and Database Access Info - created by Adam Mravnik - Jan 31

Tentative Meeting Attendance - created by Adam Mravnik - Jan 26

Home - by Luke Brisebois - Jan 25

Home - by Michael Fulton - Jan 25

Bugs - created by Michael Fulton - Jan 25

Roles - by Michael Fulton - Jan 24

Roles - by Luke Brisebois - Jan 24

Home - commented by Steven Hancock - Jan 19

Roles - by Drake Zarowny - Jan 19

Objectives - by Steven Hancock - Jan 18

TeamEffort - attached by Adam Mravnik - Jan 17

CMPT 371 - Team Effort - created by Adam Mravnik - Jan 17

Home - created by Adam Mravnik - Jan 17

# Appendix D

**Executive Summary**

The best practice being implemented for my portion of Milestone 3 is the use of Binary Mini Milestones and estimation practices. These two best practices will mainly focus on the administration and management sides of our project and our group. Both of these tools are extremely helpful in keeping projects on track. They allow for ways to monitor and estimate time requirements for tasks and the completion of these tasks. I will look at how changing to Binary Mini Milestones have changed our group dynamic and how they will help us improve production, time management and risk analysis. Using the key best practice of estimation will help estimate time and risk requirements for the rest of the project this term.

**Binary Mini Milestones**

To help better manage our team, we have implemented Binary Mini Milestones for this Milestone 3. These Binary Mini Milestones will help better define each group members tasks and specific work and will also provide an easy and accurate way in which to monitor our progress for this milestone. The Mini Milestones are as follows:

* *Drake* – Download and install Maven to become familiar with its project object model file format and get the project working in netbeans.

Evaluation: Complete Incomplete

* *Adam*– Learn and implement JMock and Log4J which both help for the best practice of rigorous risk driven testing.

Evaluation: Complete Incomplete

* *Simon*– Investigate daily build, continuous integration, and smoke tests to be implemented into the project.

Evaluation: Complete Incomplete

* *Luke*– Implement Binary Mini Milestones to set goals and tasks for each team member and look at the estimation practices to help focus on team management.

Evaluation: Complete Incomplete

* *Patrick*– Learn and Implement EMMA, a free java code coverage testing tool, for our project.

Evaluation: Complete Incomplete

* *Steven*– Learn and implement VisCad and NiCad, both of which are excellent tools for clone detection.

Evaluation: Complete Incomplete

* *Tom*– Peer Review each team members work to be compile mini reports with helpful insight into how to improve individual work.

Evaluation: Complete Incomplete

**Estimation**

For Milestone 3, Range Estimates were chosen for the numerous benefits they will provide. They explicitly differentiate best vs most likely vs worst case scenarios for each task. They will make it less likely that individual group members mistake worst case as a most likely case. Range estimates provide our client with important information regarding the uncertainty of estimates and will decrease the risk that an estimate is considered to be a commitment.

For this milestone, all estimation practices will focus specifically on this milestone and each of the previous Binary Mini Milestones that are outlined above. There are four key rules of thumb to consider for large software projects. These rules of thumb are as follows:

1. *Rule of Thirds –* requirements/design, testing and developing all require roughly the same time.
2. *Rule of Three –* if you can’t think of three circumstances in which the design might fail, you haven’t thought about it enough.
3. *Paradox Rule –* if there is no paradox that the design must resolve, you don’t understand the problem.
4. *Law of Insatiable Appetite –* the last ten percent of performance generates one-third of the cost and two-thirds of the problems.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Hours to Complete | | | | |
| Name | Item | Best Case (25%) | Most Likely Case | Worst Case  (25%) | Expected Case (50%) |
| Drake | Maven | 2 | 2.5 | 5 | 3.16 |
| Adam | JMock/Log4J | 3 | 4 | 7 | 4.67 |
| Simon | DB/CI/ST | 3 | 3.5 | 4 | 3.5 |
| Luke | Mini MS/Est | 3 | 3.5 | 4 | 3.5 |
| Patrick | EMMA | 4 | 4.5 | 5 | 4.5 |
| Steven | VisCad/Nicad | 4 | 5 | 7 | 5.33 |
| Tom | Peer Review | 2 | 3 | 5 | 3.33 |

Figure 1. Range Estimate for Milestone 3

**Because of the amount of images in the following appendixes, the appendixes have been migrated to separate PDF documents.**

# Appendix E

For more information about Team City please see AppendixE.pdf

# Appendix F

For more information about NiCad please see AppendixF.pdf

# Appendix G

For more information about VisCad please see AppendixG.pdf

# Appendix H

For more information about VisCad please see AppendixH.pdf

# Appendix J

For more information about Maven please see AppendixI.pdf

# Appendix J

**JMock - An Expressive Mock Object Library for Java - Report**

**Installation**

Download the most stable and recent Binary JAR files and Javadocs from <http://www.jmock.org/download.html>.

Extract jmock-\*.jars.zip.

Open your Java IDE and follow standard procedure for adding them to your project.

Download the most recent version of JUnit from <https://github.com/KentBeck/junit/downloads>.

Extract junit\*.zip to your project directory and add the Jar files using your Java IDE.

**Usage And Explanation**

JMock allows developers to create “Mock Objects” that simulate real objects in Java. The main benefit of this is that it can allow classes to simulate other classes and can help to avoid dependency issues when testing. A further example would be if Class A depends on Class B but Class A is what needs to be tested, then “Mock Objects” can mimic the behavior of Class B without needing to use Class B and thus resolving dependency issues.

**Basics of Integrating JMock with JUnit Tests**

1) In your unit test create a field called

private Mockery mockContext = new Mockery();

The Mockery represents the context in which the testing class exists. The context mocks out the objects that the class collaborates with and checks that they are used correctly during the test.

2) When writing your test and you have a dependency you wish to Mock you can do the following where “Object” is your actual object.

final Object myObject = mockContext.mock(Object.class);

3) Now you need to set Expectations. Expectations are what the Mock checks in your test. Below is an example from one of my own testing files, Expectations have a huge amount of commands associated with them that can be found in the JavaDoc. The easiest to use is “oneOf “and “will” which will make sure that object exists and then verify what it’s value is.

mockContext.checking(new Expectations() { {

         oneOf(response).setContentType("text/html;charset=UTF-8");

         ignoring(response);

         oneOf(request).getParameter("buttonPressed");

         will(returnValue("Add/Remove Team"));

         will(returnValue("Add/Remove User"));

         will(returnValue("Edit Rosters"));

         will(returnValue("Edit Schedule"));

         will(returnValue("Update Stats"));

         will(returnValue("Post Announcement"));

         will(returnValue(""));

         oneOf(request).getParameter("Season");

         will(returnValue(""));

         oneOf(request).getParameter("Season");

         ignoring(response).getWriter();

        }});

4) Run your JUnit tests and see if your object was Mocked correctly!

**Actual Project Usage and Experience**

JMock is a tool that allows you to apply mock objects in JUnit for test-driven development. It is particularly useful for testing things with external components and is great at speeding up Unit Tests by mocking out heavy dependencies.

JMock was used in this project mock out connections required to run some of the more important servlets. Servlets are normally a pain to unit test but by using JMock as seen in the code paste above we were able to test that the structure of servlets stay consistent.

An intereting point to note would be that JMocking servlets is a little bit counter intuitive. It is very hard to write tests before hand for a servlet in this way because one would have to know absolutely everything they were going to write and not just the logic behind it. However, in the end it proved to be an essential tool in testing the servlets in this project by checking for consistent values and structure of the code while not needing to send and receive actual HTTP requests.

In summary, JMock is a great tool. It speeds up test-driven development and allows certain aspects of code to be tested that is usually more difficult with traditional methods.

**Future Usage Ideas**

After seeing how well JMock went with servlets it will be considered using it to test JDBC SQL connections and tasks so that the database is not physically modified with entries during unit testing.