|  |
| --- |
| **Team Effort**  **Soccer Team**  **Management System**  Milestone 5 |
|  |
| **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 4

Estimation 5

Rigorous Risk-Driven Testing 6

Focused Prototypes 8

Peer Review 9

2.3 Implementation of Management Tools 10

AspectJ 10

CheckStyle 11

3. Milestone (Revised) 12

3.1 Previous Milestones 12

3.2 Current Milestone 16

3.3 Future Milestones 16

4. Risk Evaluation 17

5. Team Personnel 17

6. Milestone Activity Report 17

6.1 Work Completed 17

6.2 Git Log 17

6.3 Wiki Log 17

7. Conclusion 17

Appendix A 18

Risk Report 18

Risk Re-evaluation 20

Appendix B 21

Appendix C 25

Appendix D 28

Appendix E 31

Appendix F 31

Appendix G 31

Appendix H 31

Appendix I 31

Appendix J 31

Appendix K 31

Appendix L 31

Appendix M 32

# 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 changed marginally since the beginning of the project.

# 2. Design Artifacts

This section describes the changes made to the project, the best practises that have been implemented, and the tools that we have used or studied to help implement the best practises or help manage the project.

## 2.1 Code Improvements

An improved user interface is being introduced in Milestone 2. A new navigation bar will allow users to quickly navigate through the application’s pages. A user will quickly be able to change from adding a team to adding a player. Quick navigation allows users to perform more actions faster, with fewer clicks. The navigation bar will allow for the application to follow the *three-click rule* that suggests any user of a website/web-application should be able to find the information or perform the action that they are attempting to perform in three clicks or less.

The project’s source code has been transformed into a unified style using TXL. Using TXL all source code, no matter the author and their own styling preferences, can easily be transformed to a single style. The single style will allow for easy readability and prevent the introduction of new bugs due to poor readability when modifying code in addition to aiding the ability to review code and spot bugs that have already been placed in the source code. Team Effort will continue to modify the TXL grammar throughout the project to conform to the group’s agreed upon coding style.

Simple refactoring was used to reorganize source files for images, javascript, and cascading style sheets. The reorganization provided better directory structure and improved readability. Which with a lot of contributors to one single product can go a long way in reducing future bugs.

## 2.2 Team Management

Team management has changed both in terms of leadership and in terms of philosophy. Group members will take turns being the group leader, however, the role of the group leader has changed. The group leader will perform the tasks that are required prior to submitting the milestone such as finalizing the milestone document, running TXL on the source code, performing a MySQL dump on the database, and the actual submission of the milestone. The change to the requirements of the group leader role has occurred because of a change in philosophy. Team Effort believes consensus among all group members is extremely important and thus, instead of being lead by a dictatorship, the group will lead by building consensus. The details of the consensus building will be discussed further later in this document.

## 2.3 Versioning System

Team Effort originally planned on using SVN as its project’s versioning control system. However, over several weeks, it became apparent that SVN would not be usable by the group. The SVN repository set up by Computer Science IT was not accessible by group members through off campus computers as a result of SSL Certificate errors. After filling bug reports with Computer Science and several attempts to resolve the problem, Team Effort decided that the time spent trouble shooting could be better spent else where in the project, so we migrated from SVN version control, to GitHub. GitHub presented its own issues as no member of the group has used it before and documentation is not as extensive as SVN, however so far, there have been no issues.

## 2.4 Implementation of Best Practices

### Bug Reporting

The team’s current bug tracking system was simply recording the bug. A new system has been developed that provides a checklist of important information that must be filled out for each and every bug. Most bug reporting systems are deployed on a Web Server to provide easy access to users, however Team Effort’s bugs will be reported through their Wiki page for simplicity and to avoid overhead.

**Title**: *Example Bug*

**Priority**: *Must fix |* *Will fix opportunistically |* *Desirable, but improbable* | *Extremely improbable*

**Severity**: *Data loss or security issue | Major functionality doesn't work as specified |* *Minor functionality doesn't work as specified*

**Reproduction formula**:

*To reproduce this example bug…*

**Assigned to person**: *Someone*

**Area of the project**: *What page? What Servlet? Etc*..

**Opened by person**: *Who found this*

**Status**: *Active |* *Fixed* *| Resolved | Closed*

**Resolved as**: *Fixed | Postponed |* *Duplicate | Won't be fixed*

**Type**: *(Regular/Regression)*

**Triage**: *whether triaged, if so accepted/rejected*

### Binary Mini Milestones

Like the previous milestone we have implemented binary mini milestones for this milestone. Binary Mini Milestones are used to keep group on task and on time. It allows for monitoring task completion and helps with time estimation.

To help better manage the team, new Binary Mini Milestones have been defined for Milestone 4. 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 –* simClipse usage for our project

Evaluation: Complete Incomplete

* *Adam**–* investigate Checkstyle

Evaluation: Complete Incomplete

* *Simon**–*Continue with risk driven incremental development as well as focused prototypes

Evaluation: Complete Incomplete

* *Luke*– Continue to implement Binary Mini Milestones to set goals and tasks for each team member and continue with estimation practices

Evaluation: Complete Incomplete

* *Patrick*– Perform rigorous risk driven testing

Evaluation: Complete Incomplete

* *Steven*– investigate AspectJ and its application to our project

Evaluation: Complete Incomplete

* *Tom*– Evaluate the best practice of seeing the big picture

Evaluation: Complete Incomplete

The following is the completion status of the previous Binary Mini Milestones:

* *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 4, 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 clients 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 | simClipse | 2 | 2.5 | 4 | 2.83 |
| Adam | CheckStyle | 3 | 4 | 5 | 4 |
| Simon | RDD/FP | 3 | 3.5 | 4 | 3.5 |
| Luke | MS/Est | 3 | 3.5 | 4 | 3.5 |
| Patrick | RRDT | 4 | 4.5 | 6 | 4.83 |
| Steven | AspectJ | 4 | 5 | 6 | 5.5 |
| Tom | Big Picture | 2 | 3 | 4 | 3 |

### Rigorous Risk-Driven Testing

Testing is the process of making sure the product does what it is supposed, and does it correctly. Software testing is absolutely crucial, especially on large systems, as software is very prone to defects and problems. It is also very difficult to know what kind of problems and how many a system a software system has without doing some actually testing. The purpose of testing, however, is not just to find major defects with the system, but also to find issues like usability annoyances and generally just ensure the overall quality of the product.

There are two major implications to this:

1. Systems should be designed with tests in mind, that is no feature should be built before there is a method of testing that feature prepared.
2. Testing should focus on such features of the system that are most critical to the products success.

In order to accomplish this testing must focus on all parts of a project, not just the code. Useability and other non-code related parts of the system are very important when considering that an unusable system is a system that no one will buy. This does not just mean that the interface should be tested, but that parts of the system should be tested based on their relative importance to system operation. Another critical part of testing is documentation. Poorly documented bugs are difficult to find, reproduce, or fix and will end up costing an organization plenty of time and money to re-find, or ignore and fix later which of course is more expensive. This means that careful bug reports for every defect or quality issue should be carefully documented by the person finding the problem, and reports should be carefully analyze by the appropriate personnel and sorted by importance. It is important to note that not just the problem itself, but the method by which it was found is very important to document so that problems can be reproduced if needed, and also so that duplicate bugs do not get recorded into the system.

*Relevant Tools*

* Bugzilla (Bug Tracking)
* EMMA (Testing)
* TeamCity (Test Tracking)
* JMock (Assists Testing)
* Log4J (Tracking, Risk Assessment, etc.)
* Maven (Test Automation)
* JUnit (Testing)
* Wiki (Tracking)

Team Effort has implemented this best practice from early on. Desired features were chosen and worked on based on their importance to the system. Testing was emphasized, and unit tests were written for all Java classes. Careful bug reports were created and tracked using the wiki, so that these bugs could be found, reproduced, analyzed for importance, and eventually--fixed. In order to ensure good documentation a bug report template was created and strictly adhered to. The greatest deficiency of this best practice is that so far new things have been added without much thought towards testing. In future milestones this will be remedied and every new feature added will have a testing procedure created prior to any actual coding.

*Example*

This will be a simple example of using this best practice in regards to a known bug in this project in a manner in which it should be used on the remaining milestone:

1. Identify a known bug: Password Case Bug
2. Identify system feature: security
3. Identify critical nature of feature: severe due to the nature of security, and the ability of administrators to alter the database
4. Decide whether this bug should be fixed or triaged: due to severity and likely relative simplicity the bug should be fixed immediately
5. Design tests to run once the bug is fixed:\* Test via user interface

* Login as a known user, use the correct password with incorrect case
* Try varying the number of incorrect case characters with at least one case of all of them wrong, one of them wrong, and more than one wrong
* If at any point up to this the user is allowed in the problem is not fixed
* If the user can then log in with the correct password (correct case), then the problem shall be deemed fixed.

### 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.

### Focused Prototypes

*From Wikipedia:* "The original purpose of a prototype is to allow users of the software to evaluate developers' proposals for the design of the eventual product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions. Prototyping can also be used by end users to describe and prove requirements that developers have not considered, and that can be a key factor in the commercial relationship between developers and their clients.  Prototype software is often referred to as alpha grade, meaning it is the first version to run. Often only a few functions are implemented, the primary focus of the alpha is to have a functional base code on to which features may be added. Once alpha grade software has most of the required features integrated into it, it becomes beta software for testing of the entire software and to adjust the program to respond correctly during situations unforeseen during development."

* Minimal mockups to test (grouped) ideas
  + Examine key issues w/o assumption that using this approach
* Risk analysis e.g.
  + Prototype most challenging or highest priority questions
  + Pick best idea from each affinity group for prototyping
  + Prototype each affinity group
* Should be for throw-away use - do not use code
* Later use should be driven by open issues & decision making needs

*UI Prototypes*

* May be done by non-technical individuals
* Rapid UI prototypes can use e.g. CSS, HTML, Flash
* Can help accelerate discussion
* Help make sure everyone agreeing to same concrete look & feel

*Why Prototype?*

User may not be always be able to provide a full design requirements at the initial stage of a project. Users require multiple re-evaluations of prototypes to finalize design requirements. Prototypes are developed and evaluated by users. After the user evaluation, another prototype will be built based on feedback from users, and again the cycle returns to customer evaluation.

* Engineering mockups critical in other domains (e.g. construction)
* Identify relationships between components
* Identify risks
* Identify potential engineering savings from design changes
* Understanding interfaces between components
* Understanding testing principles

*Prototyping Process*

1. Basic requirement identification

* Any and all requirements that can be determined at earlier stages should be a part of this identification process.

1. Initial Prototype

* Initial prototype is developed based on the basic requirements

1. Review

* Customers/users should examine the prototype and provide feedback

1. Develop new prototype based on review

* The feedback is the driver now. If the feedback isn’t extensive enough development of new prototypes will be difficult.

*Prototypes*

The original development group from CMPT 370 created a series of prototypes using incremental delivery in which more features were added each time, and those included as well as the GUI were refined during each release. A series of slighty revised and modified prototypes have been released over the course of this project. Due to the requirements of the class, not a lot in terms of added functionality has been done.

*Current Prototype*

The current prototype can be viewed at <http://rtvt.usask.ca:8080/cmpt371group02/> (best viewed on Firefox).

### Peer Review

Peer review is the process in which peers review coding, documenting, and other work that is being performed by individuals. The reviewers provide feedback to the individual. Peer review is not only important for overall group performance, but also an important feedback mechanism for individuals. Milestone 3 saw an intensive and thorough peer review of every member’s participation in the project.

*Grading Scale used:*

Communication effort: /2

Thoroughness and quality of documentation: /10

Communication ability: /8

*Communication Effort:*

Members will be given 0 if nothing is provided to me in time to include it in the presentation, 1 if a group member must take effort specifically to ask them for documentation, and 2/2 if they made their documentation easy to access (e-mailed it to me or put it on the wiki) as requested.

*Thoroughness and Quality of Documentation:*

This will be an estimation of how much of the expected material the given member wrote about, and how in-depth of an explanation was given (note that giving the peer reviewer *too* in-depth of an explanation is seen as redundant, and may lose marks)

*Communication ability:*

Communication is an important skill in *any* field of work, and this is most evident when dealing with technical details and projects with lots of finicky bits and interconnected pieces. Thus, every member's ability to clearly and concisely communicate will be examined. Spelling and grammar issues may result in losing "marks" in this area (if they're bad enough).

**Appendix M** has a list of one review performed on each group member.

## 2.5 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.

### AspectJ

Aspect-oriented programming is a paradigm which allows increased modularization of code by cross-cutting of concerns. AspectJ is an extension built for IDEs such as Eclipse or Netbeans that allows users to use Java like programming to form code as aspect oriented programming (AOP). All Java programs are AspectJ programs, however, AspectJ allows for programmers to define constructs called aspects. These aspects can contain several entities that are unavailable to standard classes.

**Concerns** - Are what need to be added to the code, most often in multiple places.

**Aspects** - Encapsulates the new concepts that are being added.

**Join Point** - A *join point* is a well-defined point in the program flow. A *pointcut* picks out certain join points and values at those points. Before method calls or after a method is return would be typical join points. A piece of *advice* is code that is executed when a join point is reached. These are the dynamic parts of AspectJ.

**Advice** - The code that is to be executed at a certain point.

There are mulitple AspectJ plugins that have been developed for Netbeans. However they are no longer being actively developed. One plugin was developed for Netbeans 3.5.x, one developed for Netbeans 6.0 and another 6.7. As a result of the lack of support for Netbeans Team Effort set up a project in Eclipse to demonstrate the usefulness of the tool.

Aspect Oriented Programming has a lot of uses:

* Untangle complex concepts that are divided across multiple methods into one aspect
  + Makes code readability easier reducing maintenance costs.
  + Easier to pinpoint location for software change
  + Developers can focus on one concern
  + More reuse of code
  + Easy modifications, one aspect across multiple methods
* Simple uses
  + Easily implement logging to multiple methods.
  + Clean input data before all database writes.

Basically anytime a feature needs to be implemented across numerous methods it would be easier to use AOP compared to going to each method and adding code, which could introduce new bugs and is there for more risky.

See **Appendix** J for more information on AspectJ.

### CheckStyle

CheckStyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this tedious task. This makes it ideal for projects that want to enforce a coding standard.

The netbeans plugin would not work for Team Effort in Netbeans 7. The CheckStyle option was always greyed out despite configuring the pattern xml in Netbeans -> Preferences -> Quality. As an alternative to using Checkstyle in the IDE, Team Effort downloaded the jar file and manually, from the command line, used CheckStyle on the Soccer System source code. We obtained the Sun Standards XML for Java CheckStyle and executing it recursively on all java source files.

Although it would take far too long to bring the code to these standards as we do not adhere to the Sun Standards only important changes were made. **Appendix K** has the complete report.

CheckStyle works great and was relatively painless to use. It enforces a coding standard to such an intense degree that it makes it hard for anyone to deviate from this standard. It helps greatly in the general process improvement best practice because if there are consistent standards then there is less room for error and bugs become easier to spot.

# 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 |

**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.

*Conclusion*

* Team Effort was able to complete all of the objectives for Milestone 3.
* Team communication has improved.
* Binary milestones has allowed Team Effort to work more independently but still accomplish goals.

*Time Sheet*

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Time Spent | I worked on… |
| Luke Brisebois | Mar 3 | 2 Hr | Implementation of JUnit |
| Mar 7 | 2 Hr | Preparing for presentation |
| Mar 10 | 2 Hr | Binary MiniMilestones and Estimation practices |
| Mar 11 | 2 Hr | Individual Report for Binary Mini Milestones and Estimation practices |
| Mar 12 | 1 Hr | Preparing Best Practices slides for M3 |
| Simon Fanner | Feb 22 | 1 Hr | Fighting with GIt and Netbeans |
| Mar 11 | 2 Hr | Reading up on TeamCity  Researching Risk-driven incremental delivery best practice |
| Mar 12 | 3 Hr | Installing Team City and writing wiki page |
| Mar 12 | 1.5 Hr | Reasearching and writing up Wiki page for “Risk-Diven Incremental Delivery” |
| Mar 12/13 | 2.5 Hr | Assembling master powerpoint |
| Steven Hancock | Mar 4 | 4 Hr | Installing BugZilla and importing Bugs |
| Mar 6 | 4 Hr | Getting NiCad and VisCad to work and writing a short how-to |
| Mar 7&8 | 0.5 Hr | Putting together group mini-milestone |
| Mar 11 | 1.5 Hr | Finishing NiCad and VisCad report |
| Mar 12 | 0.5 Hr | Creating NiCad and VisCad Slides |
| Mar 13 | 2 Hr | Putting together M3 Document |
| Adam Mravnik | Feb 25 | 1 Hr | Dove into Log4J |
| Mar 6 | 2 Hr | Looked more into log4j and into jmock |
| Mar 10 | 3 Hr | Research Jmock and tried to implement with servlets |
| Mar 11 | 6 Hr | Implemented JMock testing for more complicated servlets |
| Mar 11 | 1 Hr | Wrote JMock and Log4J Wiki write-p for the handin doc |
| Mar 12 | 0.5 Hr | Wrote up presentation slides |
| Amin Shaker | N/A | N/A | Nothing Reported |
| Patrick Weckworth | Mar 7 | 0.5 Hr | Selected Tool and wrote up paragraph |
| Mar 10 | 2 Hr | Learned about EMMA |
| Mar 10 | 2 Hr | Got EMMA working in Netbeans |
| Mar 10 | 1 Hr | More or less got Emma working and wrote up wiki page |
| Mar 10 | 0.5 Hr | Did up slides |
| Tom Wetzel | Mar 12 | 2.5 Hr | Peer Review and reports |
| Mar 13 | 2 Hr | Peer Review |
| Mar 13 | 1.5 Hr | Creating Presentation |
| Drake Zarowny | Mar 5 | 3 Hr | Learning about Maven and getting basic project setup in Netbeans |
| Mar 10 | 3 Hr | Getting out project working as a Maven project |
| Mar 11 | 1.5 Hr | Writing report on Maven |
| Mar 12 | 1 Hr | Making presentation slides for Maven |

**Milestone 4**

*Objectives*

* Implementation of additional tools like Checkstyle, AspectJ, etc
* Focus on some best practices.

*Conclusion*

* Although this Milestone’s time period was short, Team Effort was able to be productive and accomplish the objectives that were set out.

*Time Sheet*

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Time Spent | I worked on… |
| Luke Brisebois | Mar 21 | 2 Hr | Writing up doc and slides for Milestone 4 |
| Simon Fanner | Mar 21 | 1.5 Hr | Writing up Focused Prototypes and modifying Risk-Driven Increments |
| Mar 22 | 1.5 Hr | Preparing Milestone 4 group master powerpoint |
| Steven Hancock | Mar 20-22 | 3 Hr | Trying to find a AspectJ Solution, writing document & slides |
| Mar 22 | 3 Hr | Putting together and submitting M4 |
| Adam Mravnik | Mar 20 | 3 Hr | Getting checkstyle to work, writing report and powerpoint slides |
| Amin Shaker | Mar 21 | 1 Hr | Writing slides and report about Defect estimation |
| Patrick Weckworth | Mar 21 | 2.5 Hr | Wrote up wiki page on best practice and did up slides |
| Tom Wetzel | Mar 21 | 3 Hr | Milestone 4 presentation |
| Drake Zarowny | Mar 21 | 1 Hr | Making a simclipse presentation (wrong simclipse apparently) |

## 3.2 Current Milestones

**Milestone 5**

*Objectives*

* Tie everything together
  + Bring all of the milestones together; link them into one defined flow.
* Finish any loose ends. Comment on all required tools and best practices.
* Evaluate team performance.
* Summarize and conclude project.

*Conclusion*

* Due to previous milestones being well done and completed thoroughly this final milestone was relatively straight forward.
* Majority of time was taken summarizing and evaluating team performance.

*Time Sheet*

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Time Spent | I worked on… |
| Luke Brisebois | - | - | - |
| Simon Fanner | - | - | - |
| - | - | - |
| Steven Hancock | - | - | - |
| - | - | - |
| Adam Mravnik | - | - | - |
| Amin Shaker | - | - | - |
| Patrick Weckworth | - | - | - |
| Tom Wetzel | - | - | - |
| Drake Zarowny | - | - | - |

# 4. Risk Report

## 4.1 Risk Identification

The following risks were identified at the beginning of the project.

*Software Requirement Risks*

* Change of requirements
  + The requirements of the software drastically changed from the beginning of the project. Team Effort removed many of the planned additions that the team 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 first milestone were some-what vague and had to be further revised for future 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 Milestones become due made allocating time to some Milestones 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 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. Using the wiki as a complimentary resource to weekly meetings helped to coordinate activities effectively.
* Not managing time affectively
  + Because there was an under-estimation of time requirements the group needed 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 helped throughout the Milestones.
* 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 was still a little ‘iffy’. In the migration to GitHub there were also difficulties as learning on the fly how to use a new version control system was not ideal.
* Lack of knowledge of tools
  + The project management tools that Team Effort would like to were all to new to the group. So time was 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 brought the possibility of introducing new errors, bugs, and unexpected results, however sufficient time was allocated to learning the new tools in and out.
* Application’s user interface not easy to use
  + This was a minor issue, however a navigation bar, would help to make the interface easier to use. For this reason, a navigation bar was implemented in Milestone 2.

*Team Risks*

* Lack of communication
  + Communication was originally a problem. However, now, Team Effort’s group members communicate efficiently and effectively.
* Scheduling conflicts
  + It has been very difficult for Team Effort to arrange any meeting times outside of the already scheduled weekly meeting. That being said, the regular meeting times have proven to be sufficient.
* 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.

## 4.2 Risk Evaluation

In light of our possible risks, the team took a number of actions throughout the course of the project to ensure maximum risk avoidance. The requirements of the Soccer System and the project were outlined as complete as possible in the first Milestone, and during subsequent Milestones Team Effort remained flexible in case new requirements, or requirement definitions changed. Any requirements that were deemed to be too expensive or impossible to meet were abandoned immediately and no more time was allocated to those requirements. Team effort always conservatively budget time to ensure time-cost over-runs were minimal. The team ensured sufficient time was allocated for testing. Through code reviews and TXL ‘pretty printing’ the software’s code, although written by multiple programmers, was all in a uniformed standard.

As a result of Team Effort’s risk analysis, most risks were avoided. The only risks that were not avoided occurred in the first or second Milestone. The risks that the team failed to manage appropriately were focused on in following Milestones and as a result all Milestones after the second saw no risk failures.

# 5. Team Personnel

Roles were redefined throughout the course of the project. Every team member during each milestone had the opportunity to take leadership roles, perform peer evaluations, develop code, implement and practice best practises, and implement software management tools. The important roles for each Milestone were identified. Each group member volunteered for one role for each Milestone. A complete Milestone by Milestone breakdown of the defined roles can be viewed in **Appendix A**.

## 5.1 Lead By Group

During Milestone one Team Effort was able to reach important decisions by forming a group consensus. As of such, Team Effort believes that having a static group leader does not conform to the reality of the group’s structure. Consensus building removes potentially unpopular, non-scrutinized, dictated decisions and ensures every important decision is thoroughly discussed and agreed upon by the entire team.

Throughout the milestones important decisions were discussed in group meetings, *Consensus Through Discussion*. As an example Team Effort had difficulties with the SVN server that was hosting the Soccer System project. The group was facing the prospect of having a project that had no version control system. During a group meeting, members of Team Effort discussed the pro’s and con’s of changing version control systems and came to a consensus during the discussion that GitHub should be used instead of SVN. As a result of consensus through discussion the entire group was able to come to a single decision through input from all group members.

*Consensus Through Editing* governed consensus building for minor decisions for Team Effort. For decisions that had to be made quickly and had little affect on the rest of the project there was not time to discuss during group meetings. For instance, take the decision to include this paragraph in this document. The author had to develop it on the fly, with little input from other group members. However, if this paragraph makes it to the final revision of this milestone it has been agreed to by consensus through editing. Any decision that is not disputed or reverted by a group member other than the author can assumed to be agreed upon by consensus. This is because of our extensive review process, every decision will be reviewed, whether that is code or documentation, by at least one peer, if that peer does not edit the decision that was made in that code or documentation it is assumed that that peer agrees with the decision made by the author. This principle is at the foundation of Wiki documentation.

# 6. Milestone Activity Report

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

## 6.1 Git Log

Please see **Appendix B**.

## 6.2 Wiki Log

Please see **Appendix C**.

# 7. Conclusion

# Appendix A

**Milestone 1**

|  |  |
| --- | --- |
| Role | Name |
| Documentation/Presentation (& Leader) | Simon |
| Documentation/Presentation | Tom |
| Documentation/Presentation | Drake |
| Coding | Amin |
| Coding | Patrick |  |
| Coding | Mike |  |
| Coding | Adam |  |
| Testing/Peer Review | Steven |  |
| Testing/Peer Review | Luke |  |

**Milestone 2**

|  |  |
| --- | --- |
| Role | Name |
| Documentation/Presentation | Steven |
| Documentation/Presentation | Luke |
| Coding | Tom |
| Coding | Adam |  |
| Testing/Peer Review | Amin |  |
| Testing/Peer Review | Simon |  |
| Testing/Peer Review | Patrick |  |
| Testing/Peer Review | Drake |  |

**Milestone 3**

|  |  |
| --- | --- |
| Tool Or Best Practise Investigation | Name |
| Peer Review | Tom |
| Daily Build/Continuous Integration/Smoke Tests | Luke |
| Jira | Amin |
| Maven | Drake |  |
| Emma | Patrick |  |
| Team City | Simon |  |
| NiCad & VisCad | Steven |  |
| Log4J and SF4J | Adam |  |

**Milestone 4**

|  |  |
| --- | --- |
| Role | Name |
| Seeing the Big Picture | Tom |
| Risk Driven Incremental Delivery & Focused Prototypes | Simon |
| Estimation Practices | Luke |
| Rigorous Risk-Driven Testing | Patrick |
| Defect Estimation | Amin |
| CheckStyle | Adam |
| AspectJ | Steven |  |
| simClipse | Drake |  |

**Milestone 5**

|  |  |
| --- | --- |
| Role | Name |
| Documentation |  |
| Mu Java |  |
| Review |  |  |

# Appendix B

commit 507e43e85ec8c023a4036b97173a169e0019cc13

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

Date: Thu Mar 22 06:26:10 2012 -0600

A start on Milestone 4

commit db1ab0bb6f104fc775784bbdd9840085a395cd0e

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

Date: Mon Mar 19 09:29:31 2012 -0600

Added the jmock jars to the project.properties file

commit b1f5be52913e58cc8e3a4ec41761f88cf4d17328

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

Date: Mon Mar 19 09:24:29 2012 -0600

Removing \_function-clones...log

commit 18842df29bf9f944ed1ce79acd07fe1f8d32d975

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

Date: Tue Mar 13 20:58:11 2012 -0600

Report for milestone 3

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 - Drake Zarowny - about 2 hours ago

SimClipse - Drake Zarowny - about 2 hours ago

Hours sheet - Simon Fanner - about 8 hours ago

Hours sheet - Thomas Wetzel - about 9 hours ago

Hours sheet - Amin Shaker - about 13 hours ago

Hours sheet - Steven Hancock - about 14 hours ago

AspectJ - Steven Hancock - about 14 hours ago

Hours sheet - Patrick Weckworth - yesterday at 11:45 PM

Roles - Thomas Wetzel - yesterday at 11:43 PM

Rigorous Risk-Driven Testing - Patrick Weckworth - yesterday at 11:34 PM

Roles - Patrick Weckworth - yesterday at 10:51 PM

CheckStyle Report - Adam Mravnik - yesterday at 10:02 PM

Home - Simon Fanner - yesterday at 04:49 PM

Focused Prototypes - Simon Fanner - yesterday at 04:47 PM

PlayerHome.png - Simon Fanner - yesterday at 04:38 PM

Login.png - Simon Fanner - yesterday at 04:38 PM

Roles - Drake Zarowny - yesterday at 03:44 PM

Roles - Simon Fanner - yesterday at 03:43 PM

Roles - Luke Brisebois - yesterday at 03:33 PM

AJ.png - Steven Hancock - Mar 20

Roles - Adam Mravnik - Mar 20

Roles - Steven Hancock - Mar 18

Meeting Minutes - Simon Fanner - Mar 14

Hours sheet - Luke Brisebois - Mar 13

VisCad - Steven Hancock - Mar 13

Mini Milestone for Milestone 3 - Simon Fanner - Mar 13

Risk-driven Incremental Delivery - Mar 13

Best Practices - Mar 13

JetBRAINS - TeamCity 7 - Simon Fanner - Mar 13

5.png - Simon Fanner - Mar 13

4.png - Simon Fanner - Mar 13

3.png - Simon Fanner - Mar 13

2.png - Simon Fanner - Mar 13

1.png - Simon Fanner - Mar 13

Hours sheet - Adam Mravnik - Mar 12

EMMA Code Coverage - Patrick Weckworth - Mar 12

Maven - Drake Zarowny - Mar 12

Mini Milestone for Milestone 3 - Thomas Wetzel - Mar 12

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 information about Team City please see AppendixE.pdf

# Appendix F

For information about NiCad please see AppendixF.pdf

# Appendix G

For information about VisCad please see AppendixG.pdf

# Appendix H

For information about VisCad please see AppendixH.pdf

# Appendix I

For information about Maven please see AppendixI.pdf

# Appendix J

For information about AspectJ please see AppendixI.pdf

# Appendix K

For information about CheckStyle please see AppendixK.pdf

# Appendix L

For information about Focused Prototypes please see AppendixILpdf

# Appendix M

**Peer Reviews**

*Tom Wetzel:*

Effort: 2/2

Thoroughness: 10/10

Communication: 6/8

Overall total: 18/20 (90%)

Tom is reasonably good at communicating in general. On the other hand, he hasn't been keeping up with my Hours Sheet, so he must improve and remember to fill that out in future. In retrospect, Tom should have tried to make a concise "marking scheme" for this milestone early and provide it to all group members before evaluation.

*Adam Mravnik:*

Effort: 2/2

Thoroughness: 10/10

Communication: 7/8

Overall total: 19/20 (95%)

Documents were placed on the wiki, and made easily accessible in good time. Documents were nicely-formatted, easy and quick to read, and still contained all the necessary information. Adam's work is great. He got through two different tools, put in plenty of effort, updated his hours sheet diligently, and wrote good documentation.

Advice: Work on organizing your thoughts before finalizing them in documents; the Experiences and Report section of your Log4J document was slightly ramble-y.

*Steven Hancock:*

Effort: 2/2

Thoroughness: 10/10

Communication: 7/8

Overall total: 19/20 (95%)

* Report included graphs, with explanations
* E-mailed me report, and put the short version on the wiki in good time
* Steven works diligently, makes good documentation, and also keeps his Hours Sheet on the wiki up to date.

Advice: Whitespace is nice when reporting data figures, for instance when you were reporting the figures from Nicad's HTML report.

Advice: Lastly, this is likely more Nicad's fault than yours, but charts and graphs are more useful when labelled for quick reference, so that you don't have to read the whole explanation and then look at the graph again to actually understand it.

*Drake Zarowny:*

Effort: 2/2

Thoroughness: 9/10

Communication: 7/8

Overall total: 18/20 (90%)

Drake’s report was e-mailed to me in good time. Drake keeps up with his Hours Sheet, and worked diligently to get Maven working despite setbacks and compatibility issues.

Advice: Some of the topics in your report weren't things that were needed (for instance, the XML Maven uses for the Project Object Model).

Advice: When writing a report adjudicating software for your company, be more decisive about whether or not you think it should used. It's hard to tell whether you think the team should use Maven despite that it is designed for larger projects, or if the team should avoid using it because it is designed for larger projects and isn't of much use to us.

*Simon Fanner:*

Effort: 1/2

Thoroughness: 9/10

Communication: 8/8

Overall total: 18/20 (90%)

* Report received after the agreed-upon time, but still in good enough time that I could include it in the presentation
* Very detailed installation guide for JetBrains, complete with pictures
* Simon gave reasons ahead of time for why his stuff would be late coming in, so I'm willing to cut him some slack; we are all busy, and he works diligently regardless.

Advice: Some of the pictures in the JetBrains documentation were redundant; the most egregious instance was having a picture of the loading screen as part of the install guide.

*Luke Brisebois*

Effort: 1/2

Thoroughness: 10/10

Communication: 7/8

Overall total: 19/20 (90%)

* Report was nicely formatted and detailed, included charts
* Time estimates seem reasonable
* Luke's work is great in general; it's very easy to extract the desired information from his report quickly.

Advice: Keep up with your hours sheet, start work on your reports earlier, and add separating lines to your tables so that it's easier to distinguish between columns.

*Patrick Weckworth:*

Effort: 2/2

Thoroughness: 10/10

Communication: 7/8

Overall total: 19/20 (95%)

* Documentation was put up on the wiki in good time; also e-mailed me to ensure that would be sufficient
* Tutorial to use EMMA was brief but very effective, good use of pictures
* Numbers and data were reported briefly, complete with datestamps
* Patrick put a bunch of effort into trying to make this tool work, and concluded decisively that EMMA is not for us, since it does not work with servlets.

Advice: Don't try to use tools that measure JUnit testing with a project that runs mainly on Java servlets. It's doomed to fail.

*Amin Shaker:*

Effort: 0/2

Thoroughness: 10/10

Communication: 6/8

Overall total: 16/20 (80%)

Amin’s documentation was not delivered on time. However, when it was delivered, the document held all necessary information and used pictures.

Advice: Since English is probably not your first language, continue working on your grammatical skills; it's understandable but could use "polishing".