CS673F14 Software Engineering
Group Project - 2
Project Proposal and Planning

Your project Logo here if any

Team Member	Role(s)	Signature	Date
Nigel Stuart Lead	• Section 1 - Overview • Section 2 contribute to similar projects • Section 3 - Project details • Section 4.b - Objectives and Priorities • Section 4.e - Schedule & Deadlines	*Nigel Stuart*	9/22/2014
Ana Beglova Configuration Leader	• section 5.c - Review process • Section 5.d Testing • section 6.b - Change & Branch control • section 6.c - code commit guidelines	*Ana Beglova*	9/23/2014
Tim Glauninger Requirements Lead	• Section 4.d - Monitor & Control mech. • Section 5.e Defect management	*Tim Glauninger*	9/25/14
Jonathan Kelley Backup Team Lead Implementation Lead	• Section 4.c - Risk Management • Section 5.b - Standards • Section 5.a - Metrics	*Jon Kelley*	10/9/14
Christopher Wright Design	• Section 6.a - Config items and tools • Section 4.a - Process model	*Christopher Wright*	10/9/14

Revision history

Version	**Author**	**Date**	**Change**
1.0	Nigel Stuart	9/11/2014	Initial team roles and responsibilities set up. Signed agreement.
1.1	Nigel Stuart	9/22/2014	Broke up document assignments across team
1.2	Ana Beglova	9/23/2014	Filled out sections assigned to me
1.3	Nigel Stuart	9/23/2014	Filled out sections assigned to me
1.4	Jon Kelley	10/9/2014	Filled out sections assigned to me
1.5	Chris Wright	10/9/2014	Filled out/updated sections assigned to me

1. Overview

This document contains the project proposal and planning for Project 2. For the semester long project the team will focus on developing an online Piano application, called Noteable. The application will be focused around helping users learn musical notes by training their ear and teaching them which sounds to associate with a particular sheet note. The application will contain tutorial levels as well as quizzes. The tutorials and quizzes will be unlocked as the prior levels are completed. As the user progresses through each level they will learn several new notes and will continue building upon each

lessons by continuing to use the notes they learned in prior chapters. As the user progresses through the lessons, they will be able to learn simple, yet popular songs, such as Happy Birthday, which will help keep the tutorials and learning process interesting.

2. Related Work

Below are some related applications that were found.

Project	Description
Virtual Keyboard	A similar virtual keyboard view, the application only provides a free play keyboard. The tutorial being developed for class will contain tutorials, which makes the application stick out from this one.
Virtual Piano	A similar virtual keyboard view, however with limited keys and also does not provide lessons and quizzes as our application will.

3. Detailed Description

This section covers functional and nonfunctional requirements of the Virtual Music Application. The application will be broken up into two main components; a front-end and a back-end service.

The front-end, a web browser, will provides users with a visual display of the musical instrument. It will be developed with HTML, CSS and JavaScript. The frontend will also utilize a sounds APIs to dynamically generate sounds and effects.

The backend will be a Java web service. The service will have a RESTful API, which will provide a mechanism to pass information to and from the frontend. The RESTful service will be written using the popular Java Spring Framework. In addition, the service will also utilize a MySQL database. The database will be used to store information about instruments as well as user recorded songs. Interaction between the database and the Java service will be performed using Java's popular Hibernate Framework.

(include some preliminary functional and nonfunctional requirements)

4. Management Plan

(For more detail, please refer to SPMP document for encounter example)

a. Process Model

The project will follow an Agile process model with three iterations over the the course of 13 weeks. The process model will not be as strict due to the fact that the group cannot meet everyday. There will be discussion of what the group has accomplished and what the group is looking to accomplish at each scrum meeting. At the end of the first iteration, planning will be done and the initial development of independent services and UI will have started. At the end of the second iteration, all backend and frontend components will be integrated together. The end of the third iteration will provide a product to meet the initial requirements set and if time permits, additional enhancement features will be added. At the end of each iteration, there will be a presentation of the work accomplished thus far. There should be evidence that some user stories have been completed to present to the customer. Also, demonstration of working software should be presented at the end of each iteration to keep the customer engaged. User stories and tasks will be assigned at the beginning of each iteration and reevaluated consistently based on various the risks assessed.

b. Objectives and Priorities

The objective of the project, within scope, is to create a virtual piano application. The application will allow users to play songs in real-time. In addition the application will provide a “Record and Play” option, which allows users to record a song, which will be persisted. The user will be able to retrieve the “Record and Play” record to be played at a later time. This feature will allow for the application to provide pre bundled “record and Play” tunes to popular songs; allowing users to see how what keys they need to press to play a popular song on their own.

c. Risk Management

See separate Team 2 Risk Management Plan document.

d. Monitoring and Controlling Mechanism

We will be checking in as a team twice a week to review what we did earlier in the week as well plan for what we will do in the up and coming week. Our meetings take place after class as well as on google hangout on sunday nights. We will also be using tools such as Github and Pivotaltracker to share our code with one another.

e. Schedule and deadlines

Below are all the milestones for the project. The milestone and features are arranged in a top down fashion and are broken up into two week cycles. Each milestones contains prerequisite features which must be completed before the next milestone can be successfully started. At the end of each milestone a retrospective will be held to demonstrate each feature. This allow the team to determine if they are on track and ready for the next sprint. Milestone 5 will provide the team time to solidify any unpredicted issues. It also considers that team members may be out of town enjoying their Thanksgiving break and/or prepare for final(s).

Delivery Schedule

Phase	Start	Finish	Status
Iteration 1			• Design Phase
Iteration 2			• Develop base framework for front & backend • Develop independent components for front and backend • Start Integrating frontend and backend connectivity
Iteration 3			• Solidify required functionality for project • Perform system testing • Make bug fixes • Final testing • Release

Feature	Developer	Iteration	Status
Design Database Schema	JON	1	COMPLETE
Design RESTful API Document	NIGEL	1	COMPLETE
Design Sound API	CHRIS	1	COMPLETE
Design Data Models	JON / NIGEL	1	COMPLETE
Training: Learn Git and promoting	ANA	1	COMPLETE
Design GUI Front-end Views	CHRIS	1	COMPLETE
Solidify Personal Development ENV	NIGEL	1	COMPLETE
Training: Coding Standards and Database versioning standards	JON	1	COMPLETE
Design Java Interfaces		1	

Solidify Personal development ENV	TEAM	2	COMPLETE
Develop base REST and Database functionality	NIGEL	2	COMPLETE
Give tutorial of Spring and Hibernate to team	NIGEL	2	COMPLETE
Investigate how to perform secure login	NIGEL	2	
Investigate Angular and how we can use it in the project			

Bug Fixes		4	
Perform code cleanup and start solidifying components		4	
Bug Fixes: continued (if need be)		5	
Perform cleanup and solidify project for completion. (continued if needed)		5	
Perform final testing to ensure application works as designed		5	
If time permits, introduce additional instrument(s).		5	

5. Quality Assurance Plan

(For more detail, please refer to SQAP document for encounter example)

a. Metrics

Software Quality Metrics

Goal: A software metric is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort to bring similar approaches to software development. Our goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

Implementation: Our team will be using Eclipse Metrics to capture the desired quantifiable measurements. This Eclipse plugin calculates various metrics during build cycles and warns the developer, via the Problems view, of 'range violations' for each metric. This allows you to stay continuously aware of the health of your code base. You may also export the metrics to HTML for public display or to CSV format for further analysis. The Eclipse metric plugin supports the following metrics:

McCabe's Cyclomatic Complexity

Efferent Couplings

Feature Envy

Lack of Cohesion in Methods

Lines Of Code in Method

Number Of Fields

Number Of Levels

Number Of Parameters

Number Of Statements

Weighted Methods Per Class

Our team will be focusing on the following metrics:

Defects – This metric will be used to determine current quality of the software by dividing resolved known defects by known defects as well as the defect density by dividing known defects by the number of lines of code

McCabe's Cyclomatic Complexity - This metric is an indication of the number of 'linear' segments in a method (i.e. sections of code with no branches) and therefore can be used to determine the number of tests required to obtain complete coverage. It can also be used to indicate the psychological complexity of a method. This is very useful for estimating integration and test efforts as well as estimating productivity for similar modules in the future.

Efferent Couplings - This metric is a measure of the number of types the class being measured 'knows' about. This includes: inheritance, interface implementation, parameter types, variable types, thrown and caught exceptions. In short, all types referred to anywhere within the source of the measured class. A large efferent coupling can indicate that a class is unfocussed and also may indicate brittleness, since it depends on the stability of all the types to which it is coupled. When used to measure couplings between packages (coming in a later release) this measure can be used together with others to calculate 'abstractness', 'stability' and 'distance from the main line'.

Lack of Cohesion in Methods - Cohesion is an important concept in OO programming. It indicates whether a class represents a single abstraction or multiple abstractions. The idea is that if a class represents more than one abstraction, it should be refactored into more than one class, each of which represents a single abstraction.

Despite its importance, it is difficult to establish a clear mechanism for measuring it. This is probably due to the fact that good abstractions have deep semantics and a class that is clearly cohesive when viewed from a semantic point of view may not be so when viewed from a purely symbolic point of view.

Lines Of Code in Method - This measure indicates the number of lines a method occupies - a line is determined by the presence of a newline character. It is a very basic measure of size and is susceptible to variation purely on the basis of different formatting styles.

Setting aside the obvious difficulties with variations produced by different formatting styles, Lines of Code counts also suffer from problems in definition due to lines of whitespace, comment lines, etc (i.e. should these be counted or not). This implementation includes all of these things in the measure. For an alternative approach that does not suffer from these problems, we will use the Number of Statements metric as well for productivity and density calculations.

Number Of Statements - This metric represents the number of statements in a method. It is a more robust measure than Lines of Code since the latter is fragile with respect to different formatting conventions. However, as in the case of Lines of Code, a large number of statements is not in itself a bad thing, it does suggest the possibility of extracting methods which gives related groups of statements a name and therefore increases the level of abstraction and deepens semantics.

All of the above mentioned metrics will continuously captured and monitored.

b. Standards

See separate Team 2 Coding Standards document.

c. Inspection/Review Process

(e.g. describe what are subject to review, when to conduct review, who do the reviews and how ?)

All code should be subject to review before new code is merged into the master branch. Code review will be conducted when someone thinks they have a working branch that is ready to be merged into master. In all cases, a team member other than the person who wrote the code should do a code review, sign off on new additions to the code base and merge the branch with the new addition to the master branch.

Git automatically sends out notifications when a pull request (request to merge a branch into master) is opened. If a new addition is not time sensitive (such as a bug in master or something that is blocking other members of the team) we should keep new pull requests open for at least 3 days or so so that everyone on the team has the chance to take a look at the code if they have time and make comments and suggestions for improving it. Code should not be merged into master until all comments are addressed.

d. Testing

(e.g. who, when and what type of testing to be performed? How to keep track of testing results?)

Each member of the team will be responsible for both automated tests and and manual integration tests for the part of the code affected by their additions before merging the code to master. The code reviewer for each feature should also ensure that there is sufficient test coverage and the tests pass as well as do a quick manual test of the new functionality. Before the end of each phase (except maybe the first few phases), everyone on the team will also black box test the code.

e. Defect Management

(e.g. describe the criteria of defect, also in terms of severity, extend, priority, etc. The tool used to management defect, actions or personnel for defect management)

6. Configuration Management Plan

(For more detail, please refer to SCMP document for encounter example)

a. Configuration items and tools

The tools related to the programming environment are:

- HTML, CSS, Javascript, jQuery, Angular js, Maven, JBoss, MySql, Java JDK, Apache, and Git

The development environment will have different requirements for the front and backend. Backend developers will be working with the Eclipse IDE and applications such as Maven to perform testing on backend services. Backend developers will also work with the Java SDK to perform RESTful services for communication with the GUI on the frontend.

Eventually, the application will be hosted as a static page on Microsoft Azure. This will simplify the integration of the front and backend environments. The group will also begin utilizing the same development environment as a virtual machine to ensure that the development stack is appropriate for all users. This stack will include the ability to access all of the related tools listed above for ease of integration between the front and backend.

b. Change management and branch management

All new code should be written in a separate feature branch. Once the code for a new feature is completed, any upstream changes from the master branch should be merged into the feature branch and the code in the feature branch should be tested and code reviewed. The code in the master branch should always be working code that can be compiled and run without crashing.

c. Code commit guidelines

As long as the head of the master branch always has working code, we won't worry about having a clean commit history. Team members should commit as often as they would like to save their work in a feature branch and we will merge branches into the master branch using merge rather than rebase.

7. References

(For more detail, please refer to encounter example in the book or the software version of the documents posted on blackboard.)

8. Glossary

Below are terms used throughout the document.

Term	Definition
REST	Representational state transfer
ORM	Object relational Mapping - Term used to database interaction via objects.