CS673F13 Software Engineering
Group Project 4 - Project Life Cycle Management
Project Proposal and Planning

Team Member	Role(s)	Signature	Date
Christian	Configuration Leader	*Christian*	9/26/2013
Vipul Agarwal	Environment and Integration Leader	*Vipul Agarwal*	09/24/2013
Vipul Kumar	Requirement Leader	*Vipul*	09/24/2013
Rachit,Yuvaraj	Design Leader	Rachit Yuvaraj	09/22/2013 09/22/2013
Alan,Manav	Implementation Leader	Alan Manav	09/25/2013 09/20/2013
Tandhy	QA Leader	Tandhy	09/21/2013

Revision history

Version	**Author**	**Date**	**Change**
1.0	**Manav**	**09/20/2013**	**Initial Draft**

1. Overview

- The intent of this project is to develop a product which would be useful in capturing the various stages of a project life cycle.
- Any software project goes through a certain defined stages right from the conceptualization phase to the execution phase. This system captures all the desired information and helps the team to be much more efficient and productive.
- This system will be developed iteratively in 3 phases corresponding to the project deadlines set in class. More functionality will be added with each working version.
- This system falls under the categories of Business Requirements Gathering System, Team Collaboration System, Document Repository, Bug Tracking System, Time Entry System, and Resource Planning System.
- The early versions are for educational purposes, as examples of software engineering practice, and as open source on which more functionality can be plugged in.
- The later versions are expected to be a commercial product marketed by software organization. The targeted audience are the software companies who are interested in tracking the life cycle of their project execution.

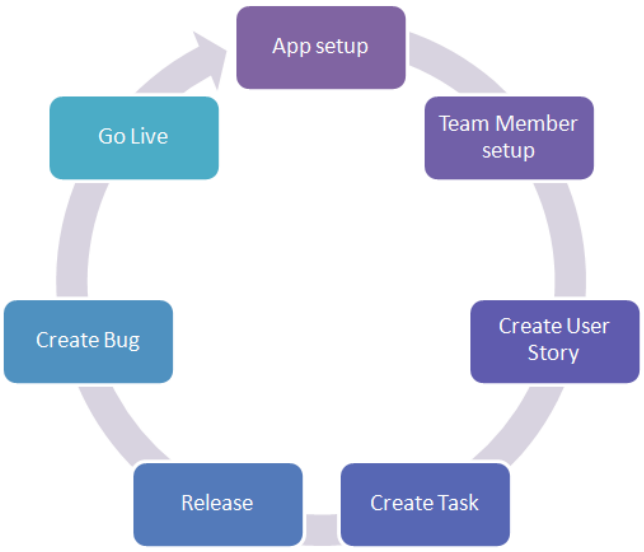
2. Related Work

- This system derives its idea from Microsoft's Team Foundation which is the Enterprise product used across organizations for their project executions.
- Pivotal Tracker has similar concepts but it suffices the purpose of just gathering Business Requirements and managing them in the form of Epic, User Stories.

3. Detailed Description

- This system is capable of capturing the Business Requirements and group them under a Release
- Role specific tasks can be assigned to team members to help them track the status of their task. Hours can be entered against each task which would be helpful in tracking the time taken by each resource for their task completion.
- All the team members have a common platform to collaborate their work. They can communicate via the chat/email features which are provided within the system.

PLM Phases



PLM System Components

- **Authentication Module –** Register team members, associate roles, authenticate and authorize based on team member set up.
- **Dashboard/Home Page –** Role-specific, customized view displaying the tasks and options assigned to each member
- **Notifications –** Captures all the notifications for which the logged in user needs to act upon.

- **User Story –** Captures functional requirements, attach documents, assigned to team member.
- **Task –** Captures the task implementation details, attach documents, assigned to team member. Ability to enter time against each task.
- **Bug –** Captures the detailed information about the system defect assigned to a developer to work on. Tagged to a specific release.
- **Workflow/Notification Engine –** Status changes will trigger notifications to corresponding resource.
- **Release Management –** Keeps track of the releases, resource allocation.

4. Management Plan

a. Process Model

The PLM project will use an iterative approach on the traditional waterfall model. The traditional phases of:

User Requirement (UR)
Software Requirements (SR)
Architectural Design (AD)
Detailed Design (DD)
Transfer (TR)

will occur sequentially for the most part. However, given that the project consists of 3 iterations, prior phases will be revisited frequently, and revisions made. Key concepts of Agile development regarding the management, organization, and structuring of a small team will be applied. Simplicity is stressed, with each iteration serving as a sprint. Potential management phase configurations will be touched upon in documentation only (if at all) given the scope of the class.

b. Objectives and Priorities

The overall objective is to create a fully functional, quality product within time and resource constraints. This will be achieved by having the PM and QA Lead in communication with the entire team at all points along the project so that issues can be taken care of in orderly fashion and timely manner. To maximize the team's efficiency and chance of success, higher priority must be given to the important tasks at hand for every iteration. Testing must occur throughout each iteration as features are implemented and tasks are completed to minimize defects cascading into later versions.

c. Risk Management

These general rules shall be followed for the entirety of the project to ensure smooth development process and to minimize the effect of potential catastrophes

- Should any problems arise, the PM should be notified immediately so the

appropriate action may be taken. This extends to anyone on the team being unavailable for any given period of time.

- The entire team must be in constant communication via weekly in-person meetings and online interaction so that everyone is on the same page.
- The scope of the project must be reasonable and locked down from an early point. This project is not strictly client-constrained as there is no one person the team is frequently in contact with. Full advantage of this should be taken to negate the potential of scope creep.
- Make sure the initial design of the system is simplistic and user-friendly. Additional functionality and complexity may be added during the development process but not at the expense of core features.

d. Monitoring and Controlling Mechanism

Monitoring of progress made is done both by the group collectively and by the PM on a weekly basis.

Weekly In-Person Meetings will take place at the end of class and run for roughly 1 hour (est. 8:30-9:30pm every Thursday). Any progress made during the previous week will be discussed and recorded. Any additional tasks for the following week will also be assigned. All relevant information is stored in the "CS673F13P4_meetingminutes" document on Google Drive immediately after the meeting taking place.

Weekly Minutes are recorded in the "CS673F13P4_weeklyreport" document on Google Drive. All team members will record the tasks they have worked on for the week, along with the number of hours spent on each task.

Pivotal Tracker and **GitHub** will be used to track tasks by priority and potential bugs/defects respectively. As tasks are completed they will be moved out of the icebox or marked as fixed.

Online Interaction between the team members, particularly with the PM will occur on an as-needed basis and serves a purpose similar to the daily meetings utilized by scrum workflows.

e. Schedule and Deadlines

The deadlines are firm and cannot be changed. If needed, human resources may be reallocated to accommodate for any scheduling issues that arise.

Date	Major Deadline	Description
09/26	Planning Presentation	Initial draft of SPMP and SQAP presented. Requirements and user stories gathered.

10/17	Iteration 1 Demo	System design complete. Basic functionality implemented.
11/07	Iteration 2 Demo	Additional functionality implemented. Most high priority tasks accomplished.
12/06	Project Due	Full functionality. All documentation finalized.

*Full schedule NYI

5. Quality Assurance Plan

a. Metrics

The following is use to maintain the progress for each task :

1. Task Complexity.
2. Time spent for member.
3. Total Defects founds.

The PLM Project will consists of 3 iterations. In each iteration, every task will have 3 properties :

- Complexity : the value of this property is a choice, the values are Easy, Moderate and Hard.
- Time Spent : the value of this property is a number of hours, starting from 0, that a member requires to finish the task.
- Total Defects : the value of this property is a number, starting from 0, that the QA team found during checking and testing.

In order to deliver a high quality project, on the first iteration the QA team will analyze development as following :

- Business Requirement : no more than 1 major and 2 minor defects regarding business requirements.

On the second iteration, QA team will analyze development as following :

- Business Requirement : no more than 2 minor defect business requirement.
- Tasks : no more than 1 major and 2 minor defect tasks for each business requirement.
- Design : no more than 1 major and 2 minor defect design for each business requirement.
- Coding : no more than 5 minor defect per business requirement.

On the last iteration, QA team will analyze development as following :

- Business Requirement : no defect found.
- Tasks : no more than 1 minor defect task for each business requirement.
- Design : no more than 1 minor defect design for each business requirement.
- Coding : no more than 2 minor defect per business requirement.

b. Standard

Documentation Standard will be based on Justin Zobel book "*Writing for Computer Science : The Art of Effective Communication*" published by Springer Verlag; ISBN: 9813083220.

Coding Standard

This section will describe the standard implementation of Coding for PLM Project :

- **File Names**
  Format for naming filename is named as the purpose for the filename. For example, filename for login purpose would be login.html. For JQuery, filename would be login.js and for JAVA, filename would be Login.java
- **File Description**
  Each file must have its description, start at the first line and commented, which will describe :
  - Filename : describe the file name.
  - Author(s) : Describe the developer who assigned to work on this file.
  - Created date : Describe date when the file was created for the first time.
  - Purpose(s) : Describe the purpose of the file
  - Feature(s) : Describe the features provide in this file

  For Example :

```
<?--
Filename : login.html
Author(s) : member1
Created date : 09-15-2013
Purpose(s) : handle login interface
Feature(s) : forgot username and forgot password features
-->

/**************************************************************
*
Filename : login.js
Author(s) : member1
Created date : 09-15-2013
Purpose(s) : handle login function
Feature(s) : forgot username and forgot password features
```

```
**************************************************************/
```

- **Function Description**

  Each function will have short description. It will describe :
  - Function Name : describe the name of the function
  - Author : describe about the developer
  - Created date : describe the date when the function was created
  - Purpose : describe the purpose of the function
  - Modified by / Date : describe if the file is modified and the developer and modification date when the change are made.
  - Modification : describe what modification are made.

  For example:

```
/**************************************************************
*
Function name : convertIntToStr
Author(s) : member 1
Created date : 09-15-2013
Purpose(s) : handle login function
Modified By / Date : member 2 / 09-24-2013
Modification :
1. change the password variable to text
**************************************************************/
```

- **Spacing**

  The purpose of spacing is for easy read by human. The following will help to address :
  - Indentation with tabs.
  - No whitespace in the end of the line or on the blank lines.
  - Lines should be no longer than 80 character.
  - `if/else/for/while/try` statement always have braces and always have multiple lines.
  - Special character operators (`!`, `++` or `--`) must not have space next to their operand.
  - Semicolon(`;`) used as a terminator line must be at the end of the line.
  - Each comma(`,`) and semicolon(`;`) must not have preceding space.
  - No filler spaces in empty constructs (`{}`, `[]`, `fn()`).
  - For each `?` and `:` in a ternary conditional must have space on both sides.
  - New line at the end of every line.

    For example :

```
if (condition) {
      statements;
} else {
      statements;
}

for ( i = 0; i < 100; i++ ) {
      object[ array[ i ] ] = functionA( i );
```

```
}

while (!condition) {
      varA++;
}

try (condition) {
      statements;
}
```

- **Switch Statements**
  When using Switch statements :
  - Use a `break` for each case.
  - Align `case` with the `switch`.

  Example:

```
switch ( condition ) {
case cond1:
    statements;
    break;
case cond2:
    statements;
    break;
default:
    statements;
}
```

- **Objects**
  When declare *objects*, there must be one line per property and indented. If the declaration is short (less than 80 character), it can be declare in a single line. Property can be quoted if they are reserved words or contain special character. Example :

```
var map = { long: 9, lat: 4, "my position": 15 };

var map = {
   long: 9,
   lat: 4,
   "my position": 15
};
```

- M**ulti-line Statements**
  - For every statement that not fit into one line, the breaks must be occur after operators and follow by indented to distinguish them for the body. Example :

```
vars textMsg = "<p>Error message is " + errMsg +
      ". Please contact Our support immediately!"
```

  - For conditional statements that is not fit into one line, the breaks must occur after the operator and be indented to distinguish them from the body. Example :

```
if ( Condition1 && Condition2 && Condition3 &&
```

```
            Condition4 ) {
            statements;
    }
```

- **Assignments**
  Assignment in declarations must have in each line. Declaration with no assignment must be listed together at the start of declaration. Example :
  ```
  var x, y, z,
  sumA = true,
  sumB = false;
  ```
- **Comments**
  Comment must be on its own line. single line comment start with `//` and multi-line comment can use either `//` or `/*...*/`. Short comment can be put at the end of the line(refer to the length limit). Example :
  ```
  // we use textMsg to hold the text that will be send to client
  // when error occurs.

  /*
  we use textMsg to hold the text that will be send to client
  when error occurs.
  */
  vars textMsg = "<p>Error message is " + errMsg + // errMsg = 2
  ". Please contact Our support immediately!"
  ```

c. Inspection/Review Process

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

Inspection are use to keep the quality of the project, improve manageability and productivity to the project development process. Inspection will be divided into 2 parts :

1. Self Inspection
   Self Inspection will be conducted by developer prior submit to QA team. The inspection will include :
   - Check whether the task(s) is completed or not
   - Check Defect for each task(s)
   - Review the Coding Standard
2. Team Inspection
   Team Inspection will be conducted by QA team after the developer submit the code. The inspection process will include :
   - Check the code and compare with the task(s)
   - Check Defect for each task(s) and file Defect report if found.
   - Review the Coding Standard

d. Testing

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

QA team will be responsible to conduct Testing process. It consists :

1. Unit Testing
   PLM use Java for the middle-tier, and therefore JUnit will be used for Unit Testing.
2. Web Testing.
   Front-end of PLM use Web-based and therefore third party software, Selenium, will be used for Web Testing.
3. Database Testing.
   Back-end of PLM use Mysql and with the third party software, DBUnit, will testing Database functionality.

QA team will keep the Unit Testing, Web Testing and Database Testing record on each testing process.

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)

For Defect Management, QA team will use Github Issue Tracking. After Testing or Team Inspection, QA Team will file a Defect Report using Github Issue Tracking System. QA Team will need to fill several data :

1. Title
2. Assigned to
3. Description
4. Defect Data : Severity, Priority, Defect Type and Code Defect Type.

To recognize defect , the following are the guidance :

1. Severity
   The value of severity is as following :
   - Major : if task(s) requirement are not satisfied
   - Trivial : will not affect the execution
   - Minor : not Major or Trivial
2. Priority
   The value of Priority refer to the follow-up action need to be taken, is as following :
   - Urgent : the defect must be follow-up immediately.
   - High
   - Medium
   - Low

3. Defect Type
   The value of Defect Type is the following :
   - Bug
   - Duplicate
   - Enhancement
   - Invalid
   - Question
   - Wontfix
4. Code Defect Type
   The value of Code Defect Type is the following :
   - Syntax
   - Logical
   - Data

6. Configuration Management Plan

a. Configuration items and tools

All configuration items will be stored in a git repository hosted on GitHub. The following configuration items are required:

- Documentation directory
  - Planning documents
  - Code documentation
  - User level documentation
- Front-end code directory
  - User interface code
- Back-end code directory
  - Services code
- Database code directory
  - Database create scripts
  - Sample insert scripts

Only essential files required for QA, deployment, or understanding (documentation) should be added to the repository. In general, binaries and temporary files should not be added. Binary documentation is allowed in the repository but the plain text source documents should accompany them.

b. Change management and branch management

The general outline is as follows:

- The master branch should be at demo quality at all times.
- Unstable code can be merged into a development branch.
- New features should go in their own branches, which will at some point be merged into the development branch.

- Approximately every three weeks, the development branch will be merged back to the master branch to prepare for the presentation.

This system should allow the team to always have a state available for a demo while continuing to add features in feature branches and fixing bugs in the development branch. To begin, one person should commit the basic layout of the project. From there, each member can begin working on feature branches based on that initial commit. At some point, a feature will be complete and can be merged back to the development branch. QA may want to review the code before merging to minimize bugs creeping into the development branch. At the end of a development iteration, the development branch will be merged back to the master branch to prepare for the presentation.

c. Code commit guidelines

Each commit should contain as few independent changes as possible and should have a descriptive commit message. Code that produces validation or build errors should not be pushed to GitHub. Broken code may be committed to local repositories but the additional commits required to produce buildable code must also be present in the repository before it is pushed to GitHub.

7. References

(For more detail, please refer to encounter example in the book or the software version of the documents posted on blackboard.)
http://contribute.jquery.org/style-guide/js/ Coding Standard

8. Glossary