**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, Christian, Alan | 09/20/2013 | Initial Draft |
| 1.1 | Tandhy | 09/20/2013 | Added SQAP Section |
| 1.2 | Tandhy | 09/30/2013 | Revise SQAP : Metric |
| 1.3 | Christian | 10/16/2013 | Revise SCMP Branch Management |
| 2.0 | Alan | 10/17/13 | Requirements first draft, Management Process revision, overall editing pass |
| 3.0 | Alan | 11/09/13 | Iteration 2 Requirements Rework |
| Final | Alan | 12/08/13 | Iteration 3 Final Update |

[Overview](#h.87t9hln2vjz0)

[Related Work](#h.mps353x5ezyl)

[Detailed Description](#h.fg3z0hpd4q9v)

[Management Plan](#h.ds8oyr75pnh1)

[Process Model](#h.27177f40uci)

[Risk Management](#h.a4oqwntk3mw)

[Monitoring and Controlling Mechanism](#h.ywdoc2clc9yt)

[Schedule and deadline](#h.tadq5mb0pici)

[Quality Assurance Plan](#h.72e1f4uawy2r)

[Metrics](#h.b2haznn3yyz2)

[Standard](#h.vc72k6dweldv)

[Inspection/Review Process](#h.f1c69ifi68h7)

[Testing](#h.r5d5mhtlf0kq)

[Defect Management](#h.54a4wuncjg1c)

[Process improvement process](#h.jhct37ebxxpn)

[Configuration Management Plan](#h.hw41vg4ykxen)

[Configuration items and tools](#h.bwlb4d4vdox2)

[Code commit guidelines](#h.yyauft6zr9hw)

[References](#h.8mva2050iy7t)

[Glossary](#h.ty3i2nqffhtc)

# 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 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.
* The project is meant for educational purposes, as an example of software engineering practice, and as an open source on which more functionality can be plugged in. The targeted audience is any student enrolled in any software engineering course who is interested in tracking the life cycle of their project execution.

# 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 in regards to gathering Business Requirements and managing them in the form of Epics and User Stories. The PLM will include a simplified version of what PT offers, amongst other features.

# Detailed Description

The concept of roles plays an important part in the overall PLM system. The two roles available to users of the system are developer and project leader. A generic role of “user” is assigned to anyone who is registered in the system but not currently assigned to any proect. Functional requirements are of two types, general and role-specific.

**3.1 General Functional Requirements**

* Users register themselves under a username and password which are stored securely on a server
* The system will allow registered users to log in given correct credentials
* Upon logging in, the system displays a view specific to the user’s role (only if they are assigned to a project)
* A registered user may create a new project, in turn making them the PL for said project (for the purpose of our project, the PL is also a developer)
* As long as a user is assigned to the project, they will be able to make comments on any given user story included in that project

**3.1.3 Project Leader Requirements**

* PLs are granted access to any and all projects for which they are a leader
* PLs add users to a project, which will set the role of that user to developer
* PLs create and edit user stories and set someone to be its “owner”
* PLs add tasks to user stories and assign them to a developer (or themselves)
* PLs can change the status of an user story or task to complete, pending, or any other available status
* PLs can create a release for a project by providing a name and a start/end date; releases cannot overlap with one another; information will be dynamically pulled so that a new release will not be able to be created in any other release’s time range

**3.1.4 Developer Requirements**

* Developers can view/ update tasks which have been assigned to them
* Developers can “attach” deliverables to a user story or task via comments

**3.2 Non-Functional Requirements**

1. As a web-based solution, the PLM system should run on all major web browsers.

Compatibility with older software is crucial as the assumption that all end-users of the product are running the most up-to-date version of a browser cannot be made. Any updates to the system must retain backwards compatibility functionality as the system is useless if users cannot access existing data.

1. The service’s uptime should be as close to 99% as possible.

100% uptime on a web-based service is unreasonable, but close to perfect uptime is very important to the usefulness of the product and client satisfaction.

1. System response time should be less than one second.

As a system that is tied to a SQL database, collected data must be kept tidy and cataloged correctly to maintain system speed. Any inconsistencies in schema/function design could cause the system to slow to a crawl over time. Maintenance and uptime must be done to the back end to ensure a good user experience.

1. Standards for internet security will be followed. Encryption methods for any function that requires it will be used. Local data will be deleted on an as-needed basis to prevent any unauthorized access.

SQL injections and other vulnerabilities must be accounted for. Due to the nature of the information handled by the service, it is paramount that no data is leaked or deleted due to inadequate security measures.

1. Non-relevant data will not be stored in the system. Automatic cleaning of database records will occur at set intervals unless explicitly stated.

Regardless of whether it is a registered user’s personal information or extraneous information concerning a project, unnecessary information will not be stored in an attempt to reduce the amount of harm done in the event where a security breach occurs. By its nature, the PLM system deals with sensitive data. The system’s primary goal is to facilitate easy tracking of project’s process, not long term data storage. Any latent, old data may be purged from the system (or archived in some other location/service) after an as-of-yet undetermined period of time. This is for security and performance reasons.

1. Sensitive data should be backed up on a regular basis.

Data loss prevention is as important as security to a web service. Customer satisfaction is based almost solely on the data stored in the system being persistent. Data back-ups will ideally be located across many sites to minimize the risk of physical data loss.

# Management Plan

## Process Model

The PLM project does not strictly adhere to one development methodology. Rather, concepts from an iterative approach on the traditional waterfall model consisting of the traditional phases:

User Requirement (UR)

Software Requirements (SR)

Architectural Design (AD)

Detailed Design (DD)

Transfer (TR)

were used. Given that the project consists of 3 iterations, prior phases will be revisited frequently, and revisions made. Additionally, 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.

## 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 most important tasks at hand for any given iteration. Testing must occur throughout all iterations as features are implemented and tasks are completed to minimize defects cascading into later versions. Individual goals for the iterations are as follows:

* Iteration 1: Basic functionality implemented. User login and registration system with proper authentication and authorization checks. General UI and design decided upon. Initial integration between browser-service-database complete. Database classes and attributes decided upon. Learning of new technologies should occur mostly during this iteration.
* Iteration 2: Implement more features. Apply any necessary changes and bug fixes from iteration 1. Due to issues that arose in iteration 1, we had to revisit the requirements phase and do a complete overhaul of what the final product should be. As a result, we decided to remove many of the roles we initially planned for. Additionally, features such as bugs and notifications were also cut from the project.
* Iteration 3: Achieve full functionality while working on making the product as polished as possible. UI/styling cleanup. Iteration 3 was mostly for polishing existing pages and adding the necessary validation checks to reduce the number of bugs/errors on the database side.

## 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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Risk | Impact | Mitigation Plan | Reason |
| 1 | Lack of Communication | 9 | Set up methods to easier facilitate communication between team members (Google Group, Skype, etc…) | Most team members have work or other commitments such that in-person meetings are tough to organize |
| 2 | Inexperience with HTML | 2 | HTML as the basis of the project is a relatively easy language to pick up and learn. Seeing demo code and examples will be sufficient in most cases | Members were given roles, but it might not have been what they were familiar with (either by choice – want to learn something new, or defaulted to them) |
| 3 | Inexperience w/ JQuery & JSON (services-related) | 6 | Everyone is coding must read up on the languages. Higher impact earlier on in the project, will lessen as time goes on | People’s backgrounds vary greatly. Common ground in terms of coding language is not agreed upon. Members will have to learn while on the project. |
| 4 | Inexperience with Github/Pivotal Tracker | 3 | Have someone who has used the service before explain it to the others | Have not worked with a version control system or tracker before |
| 5 | Inexperience with Kendo framework | 6 | Member demonstration for the others; delegate work to those who are more familiar - in the end, some will avoid while others will conquer | Many members have not used any sort of framework before; flexibility when choosing which to use, but steep learning curve for most |
| 6 | Losing/Gaining team members | 4 | There is no easy solution to this problem. New members will have to pick everything up as they go along. Lost member’s tasks must be divided amongst remaining members | Due to personal commitments, the members on the team are subject to change. Luckily all changes occurred relatively early on so recovering wasn’t too bad |
| 7 | Team Members Unavailable | 8 | Tasks assigned to unavailable team member will be distributed amongst the rest of the team; member must catch up with everyone once they have time | Due to personal commitments, certain team members may not be available for a week or two during the project |
| 8 | Scope Creep / Time Management | 10 | Features may have to be cut from the final product due to time restraints and initial estimates being wrong. Due to the nature of the class, conquer methods are unlikely to work; core/major functionality must be implemented first while bells and whistles left for the end if time allows for it | List of functionality for the product to be considered “fully functional” determined in planning phase, but may revise due to unforeseen circumstances |
| 9 | Uncertainty about one’s assigned tasks | 4 | Be in constant communication with the team at the weekly meetings; restate goals and priorities for between every iteration so milestones are clear | A lot of things are said at the weekly meetings, but one might not pick up on everything in the moment, must leave early, etc… |
| 10 | Inconsistency in Implementation Schedule | 8 | Team members must try their best to be coding throughout each iteration and not put everything off until the last week of each iteration | The first and second weeks are usually reserved for discussion and planning, while coding and QA is left until the last week |
| 11 | Incorrect Requirements Phase | 10 | Group meetings and Skype conversation to revisit what features are essential/core to a working product; reanalyze what is practically deliverable by the end of the class; implement and do more work in later iterations to make up for lost time | After iteration 1 it was brought to our attention that our definition of the project was beyond of the scope of the class and overly complicated with high coupling. It was necessary to break the project down and rebuild it as something that would be completed by iteration 3 |

## 

## 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.

## 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. |
| 10/03 | Iteration 1 Design and Planning Phase | Organization/communication improvements via Google Groups, upload to GitHub; framework discussion; write services required for user stories; design discussion |
| 10/10 | Finalize Iteration 1 Features | Database schema update; integrate all html pages with web services front-to-back end; apply CSS and Kendo framework to all pages; use case and design diagrams; integration tutorial |
| 10/17 | Iteration 1 Demo | Basic system design complete; Login, registration, dashboard and add role functionality implemented; SPMP and major documentation updated |
| 10/24 | Iteration 2 Design and Planning Phase | Complete revision of business requirements analysis, remove and add features as needed, decide on user stories to implement; revise database schema to accommodate changes; continue bug testing and styling of system |
| 10/31 | Iteration 2 Implementation | Implement: Create Project, Create Iteration, Task Management, User Story Management, Dashboard features, polish iteration 1 features |
| 11/07 | Iteration 2 Demo | Iteration 2 Testing Done, Metrics Gathered, Demo presented, Planning for final iteration features |
| 11/14 | Iteration 3 Implementation | Polish existing user stories + add functionality to pages, bug fixing, implement manage roles/team members feature, add validation checks to all pages so that invalid inputs won’t be allowed |
| 12/06 | Project Due | Full functionality. Final presentation. |

# 

# Quality Assurance Plan

## Metrics

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

1. Total Line of Code (LOC).
2. Total of functions.
3. Man hours.
4. Total Defects founds.

The PLM Project will consist of 3 iterations. In each iteration, all tasks will have 4 properties and QA team will evaluate and keep its track:

* Total Line of Code: the value of this property is a number, which reflect the total implemented line of coding on each task.
* Total of functions: the value of this property is a number of functions, which define in each task.
* Man hours: the value of this property is a number of hours, starting from 0, which a member requires to finish the task.
* Total Defects: the value of this property is a number, starting from 0, which 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 3 major and 3 minor defects regarding business requirement.
* Tasks: no more than 3 major and 3 minor defects for each business requirement.
* Design: no more than 3 major and 3 minor defects for each design requirement.

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.

## Standard

**Document Standard**

PLM document standard will be based the following document:

* SPMP
* SQAP
* Software Design Document

**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 modifications 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 characters.
* If/else/for/while/try statements 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 :

var 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 sent to client when error occurs.

\*/

var textMsg = “<p>Error message is “ + errMsg + // errMsg = 2

“. Please contact Our support immediately!”

## Inspection/Review Process

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

Inspection is used 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

1. 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

## 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.

1. Web Testing

Front-end of PLM is Web-based and therefore a third party software, Selenium, will be used for Web Testing.

1. Database Testing

Back-end of PLM uses MySQL. The third party software, DBUnit, will test Database functionality.

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

## 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

1. 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

1. Defect Type

The value of Defect Type is the following :

* Bug
* Duplicate
* Enhancement
* Invalid
* Question
* Wontfix

1. Code Defect Type

The value of Code Defect Type is the following :

* Syntax
* Logical
* Data

# Configuration Management Plan

## 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.

## 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 to the master branch and create a development branch. From there, each member can begin working on feature branches based on the new development branch. Each feature should have its own branch and each branch should be given a short yet descriptive name representing the feature. 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.

## 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.

# 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

# Glossary