Attention:
HOA DAM
CSCI311
7 April 2014

Software Plan (Detail)

Table of Contents

Version History

Date	Version	Comment	Author
6th April	0.1	First draft	James Wilson
7th April	0.2	Second draft + Section 5	James Wilson
8th April	0.3	Final + Added Sections 1 & 2	James Wilson

Introduction

This report details the implementation plan of Team Awesome's new software product called PyTag. This software will allow for the developers and contributors to search the Python code base, issue database and mailing lists from a central Dashboard.

Abbreviations

TODO: Insert document-specific abbreviations.

References

TODO: Insert documents that this report references.

1. Scope

1.1. System Overview

This system will be comprised of two major components - client side and server side - with several modules making up the server-side component.

4. General Software Activities

4.1. Software Development Process

The chosen development method for this product will be the ‘feature-driven’ method, a derivative of the Agile development method. This method is an iterative-based approach to the development of the client software, whereby each feature is designed, planned, tested and then integrated into the main program. Each cycle is an ‘iteration’ and is detailed in the Gantt chart.

4.2. General Plans for Software Development

4.2.1. Software Development Methods

Feature-driven development requires extensive testing of all implementations of the various features when being imported into the main code base. The tools that will be used in this process include:

- **Git** - the use of a Git repository to share code amongst the team, so that all members can modify files individually and then commit them so others may test their changes
- **Travis-CI -** a continuous building and testing system Travis-CI will allow for each commit of a feature (during an iteration) to be tested against a set of pre-defined units. This will mean commits that fail to meet a build status of ‘pass’ will not be allowed in the main branch.
- …

4.2.2. Standards for Software Products

The final deliverable - the ‘PyTag’ software - is comprised of two parts: client and server.

The client (or end-user) component will be made to conform to HTML5, JavaScript and CSS3 standards (commonly known as ‘web’ standards). This will ensure maximum cross-platform compatibility without a loss in function.

As the client has not specified a desired language for the development and running of the software, Ruby on Rails has been chosen by Team Awesome as the platform that will power the server-side component of the software.

To ensure that final software product is delivered on time, and with the features requested from the client, a standard will be applied to all code submitted. All source files will be documented so that future development may occur with minimal cost and training / down time.

4.2.3. Reusable Software Products

The application shall take full advantage of existing solutions (known as ‘Ruby Gems’) publicly available for commercial use if such a solution is found to solve a minor or complex part of the software.

A minimum of one (1) day will be spent on testing and attempting to implement Ruby Gems within the existing codebase.

4.2.4. Handling of Critical Requirements

Critical requirements of the project, as determined in the “Software Requirements Sheet”, will be guaranteed as deliverable under the following:

4.2.4.1. Security Assurances

Team Awesome shall put in place a user-account system analysing and transforming data to ensure that only developers have access to the code and that actions which may result in data deletion or modification require appropriate access levels and authority before proceeding.

4.2.4.2. Privacy Assurance

Team Awesome shall undertake utmost care and diligence when information contained with given Archive is not accidentally disseminated to the public from PyTag without prior approval of the author or owner of the given piece of information

4.2.4.3. Other Assurances

Team Awesome shall ensure that any code taken from a public domain be correctly referenced and licences shall remain intact, to negate (to the best of their ability) any legal actions that may be taken against the Client.

4.2.5. Hardware Resources

The final deployment of the Software will require a server running nginx, Ruby, MySQL all running under a Linux OS.

4.2.6. Recording Rationale

Team Awesome shall record meeting minutes, work diaries, IM conversations, emails and other information during the development of PyTag. Key Decisions made in the project shall be voted on by the Team and the results recorded in the meeting minutes. All arguments for and against will be noted for reference if a dispute occurs in the future.

4.2.7. Client Review

The Client will have the ability to, at any stage of PyTag's development, arrange a meeting to discuss progress and review functionality of the Software. The Client will be the authorised party to make changes to the SRS, provided Team Awesome inform them of their ability to deliver the proposed changes after a vote is performed and noted.

5. Software Development Activities

5.1. Project Planning and Oversight

5.1.1. Software Development Planning

Team Awesome shall do something.

5.1.2. Computer Software Configuration Item (CSCI) Test Planning

Each configuration item shall be tested locally on the developers machine before being submitted to the main code base. Upon submission, the changed CSCI will be tested against an objective, independent environment to ensure no code breakages.

5.1.3. System Test Planning

Team Awesome shall 'stress test' the server environment to ensure no resource constraint regressions are introduced so as to utilise the available hardware resources with peak efficiency. System Testing will be completed after each milestone.

5.1.4. Software Installation Planning

The Software will require web-hosting space meeting the requirements specified in [XXX].

5.1.5. Software Transition Planning

As there is no current software in place, a software transition from a previous software application to PyTag is not required. User training will be required for the Client to be able to use and correctly understand the system.

5.1.6. Management Review

The Team shall meet with the Client each week, during the allotted CSCI311 tutorial time. Should the need arise for an extra meeting, a place and time will be arranged with the Client at a time that suits all stakeholders.

5.2. Software Development Environment

5.2.1. Software Engineering Environment (SEE)

The software engineering environment will consist of:

- Any text-editor of choice by the Developer,
- Ruby on Rails installations via ‘Homebrew’ on OS X and the provided RoR executable on Windows
- MySQL executables on both OS X and Windows

These will be established once design of the system has been finalised, controlled by each developer on their local machine and maintained by the group to ensure consistency across all platforms.

5.2.2. Software Test Environment (STE)

The software test environment will consist of numerous ‘bots’ that will test specific scenarios with each CSCI / module developed. These bots will be created by all members of the team to thoroughly check for any leaks, corrupted data and high resource utilisation.

To establish a bot, each member will need to sign off on the bot that will run under the Travis-CI environment to ensure a complete understanding of the intended functionality of the CSCI / module. Control will be handled by all members to ensure rogue bots are either terminated early with their activities analysed. Maintenance will be handled by James Wilson as the project manager to ensure that testing provides an accurate insight into the development of the software.

5.2.3. Software Development Library (SDL)

The software development library used will include original Developer content with open-source, community-provided Ruby Gems.

5.2.4. Software Development Files (SDF)

The software development files will be maintained by all members of the group, with each member committing to their own copy of the main code base any changes. No establishment restrictions are required for this.

5.2.5. Non-deliverable Software

There are no non-deliverable software requirements for this project.

5.3. System Requirements Analysis

5.3.1. Analysis of User Input

To verify that there is understand of the Interface, at each meeting, Team Awesome shall update the Client on the progress of the Interface and ask them to interact with the Interface, whilst noting how the client uses the software.

Eliciting feedback from the Client will then commence, with any suggestions and comments from the Client to be noted and used to align the Interface with the Clients requests.

5.3.2. Operational Concept

The operational concept behind the Software will be derived from the Software Requirements Specification and the Feasibility Study. Refer to this document for the operational concept.

5.3.3. System Requirements

The requirements for this system have been defined in the supporting “Software Requirements Specification” document. Please refer to this document to examine System Requirements.

5.4. System Design

5.4.1. System-wide design decisions

Participating in system-wide design decisions will be conducted with a vote by the group, with all results noted in meeting minutes.

If the proposal in question potentially impacts SRS-defined requirements, the proposal will be sent to the Client for their input with the Client’s response to determine if the proposal be accepted.

5.4.2. System architectural design

Participating in system architectural design will be conducted with a vote by the group, with all results noted in meeting minutes. As the architecture is not Client-visible, these changes will be voted on by the Group.

The Model-View-Controller architecture has been chosen for this project.

5.5. Software Requirements Analysis

The software requirements analysis will occur before implementation of the specific requirement to ensure the requirement can be implemented in a timely manner.

5.6. Software Design

5.6.1. CSCI-wide design decisions

Participating in CSCI-wide design decisions will be conducted with a vote by the group, with all results noted in meeting minutes.

5.6.2. CSCI architectural design

Participating in CSCI architectural design decisions will be conducted with a vote by the group, with all results noted in meeting minutes.

5.6.3. CSCI detail design

Participating in CSCI details design decisions will be conducted with a vote by the group, with all results noted in meeting minutes.

5.7. Software implementation and unit testing

5.7.1. Software Implementation

Implementation of the Software will occur on a new web server that does not have any prior data on it. Team Awesome will compress and obfuscate the production code to ensure source code and integrity.

5.7.2. Preparing for Unit Testing

Unit Testing preparations will be completed by Team Awesome designing the bots to test all aspects of finalised features to the main codebase.

5.7.3. Performing Unit Testing

Unit Testing will be performed automatically with every Git commit by the Travis-CI system. The result will be appended to the commit to alert the Developer of the bot's actions.

5.7.4. Revision and Re-testing

If finalised modules result in a bot reporting an error, the Team will analyse the changes to ensure that a new (or the current) bot needs to be created (or modified) to be to ensure that the test outputs being examined are the outputs Team Awesome expects.

5.7.5. Analysing and Recording Results

The results will be automatically logged by Travis-CI. If a test succeeds, the team will ensure outputs match the expected outputs to ensure no data corruption. If the test fails, the team will analyse the results (as per 5.7.4).

5.8. Unit Integration and testing

5.8.1. Preparing for unit integration and testing

The Team will fork the main codebase before commencing development to ensure local develop can occur, independent of there teams. When the CSCI is ready to be added, a Pull Request will be issued from the Developer to the main code base indication that the commit is ready.

Travis-CI will test the build and attach the result to the Pull Request.

5.8.2. Performing unit integration and testing

The integration of the request will automatically occur on the GitHub website by the Change Control Manager.

5.8.3. Revision and Re-testing

If a commit breaks the code base, the Team will vote on the direction that needs to be taken. If a revision needs to occur, the commit will be rolled back to the previous state and a post-mortem will occur to examine if the testing or code failed and caused the erroneous Pull Request to be accepted and merged.

5.8.4. Analysing and Recording Results

Results will be logged on the Github commit history as well as work diaries and milestone tracking tools used by the Team.

5.9. Preparing for software use

5.9.1. Preparing the executable software

The server component does not compile to a single executable. The source files will be obfuscated and compressed to insure the smallest disk space foot print, as well as optimisations for computationally-intensive tasks.

5.9.2. Preparing user manuals

The Team shall, upon completion of the Interface and server components, create and document all functionality of the system, to ensure that the Client can use the Software with a complete help and reference library available to them.

5.9.3. Delivery of user manuals

The Team will deliver user manuals in a complete website, with static HTML, to allow for the Client full search and bookmarking capabilities.

5.10. Software Product Evaluation

5.10.1. In-process and final software product evaluations

Team Awesome shall, after the completion of each milestone, inform and meet with the client to evaluate the Interface to ensure the SRS-defined requirements are consistent with the Client's expectations. If the Client requires it, Team Awesome shall also meet to examine server-components to ensure that server processes meet the needs of the Client.

5.10.2. Software Product Evaluation Records

After each evaluation, the records will append to a 'Milestone Evaluation Diary' that will be maintained by the Project Manager. This diary will be presented along with all deliverables at the conclusion of the Project.

5.11. Corrective Action

5.11.1. Technical and Management Reviews

Team Awesome shall endeavour to cohesively solve any technical and non-technical matters that may arise during development. If no resolution can be reached, the Client shall be informed if the problem impacts an software requirement.

If a review of the current direction of the project is required. a set of proposals will be examined to ensure the project remains in scope and will be delivered on time.

5.12. Other Development Activities

5.12.1. Risk Management

Please see the supporting “Risk Management” document for all associated risks and mitigation strategies for this project

5.12.2. Software Indicators

Objectives and milestone completion will be used as indicators that the Project is on track to be on budget and on time.

6. Project Resources, Organisation and Schedule

6.1. Assigned Roles

Please refer to the Feasibility Study for the assigned roles in the Project.