Quality Plan

for the

Hotspotter Bug Prediction Software

CS 425 / CS 499 Senior Project

by

Nathan Reinhardt

Spencer Smith

Dylan Williams

of

Team HotSpotter

QUALITY-PLAN

Revision 1.1

As Of: 11 October 2015

Change Log:

Revision	**Change Note(s)**
1.0	• Initial release
1.1	• Added use of Jira and test details

1 INTRODUCTION

This document is the quality plan for the Hotspotter Bug Prediction Software developed by Team HotSpotter.

1.1 PURPOSE

The purpose of this Quality Plan is to identify and represent the methods to be taken by the team to ensure that the quality of the product fulfills all client and course-specific requirements as outlined in the project specification

2 MANAGEMENT

In order to ensure proper division of labor, development pace, customer communication, and quality assurance, the Scrum-based *Semi-Agile Software Engineering* (SAGE) process will be utilized.

2.1 ROLES AND RESPONSIBILITIES

The SAGE Process dictates at least three roles to be filled. A brief summary of the role as well as the team member assigned is as follows:

- *Scrum Master*
  Nathan Reinhardt
  The Scrum Master will lead team meetings and ensure that the SAGE process specification is followed.

- *Customer Proxy*
  Spencer Smith
  The Customer Proxy will be responsible for establishing contact with the client and ensuring that the team maintains the same understanding and vision as the client at all times.

- *Quality Assurance Manager*
  Dylan Williams
  The Quality Assurance Manager will be responsible for maintaining the team's best practices throughout planning and development and ensure that the Quality Plan as described in this document is properly followed.

As well as fulfilling the responsibilities outlined for each role above, each team member is expected to stand in as any auxiliary roles that may arise throughout development.

2.2 TEAM MEETINGS

All members of the team will meet regularly once a week outside of the scheduled CS 425 / CS 499 course time. As two-week sprints will be used in accordance with SAGE, the first weekly meeting of a sprint will attend to any review and retrospection of the previous sprint and cover all necessary planning for the newly beginning sprint. This includes selecting tasks from the product backlog to be assigned and identifying and reevaluating any unfinished tasks from the previous sprint. The second meeting of a sprint will occur halfway through the current sprint and will be to discuss the progress each member has made in their assigned tasks. If any unexpected roadblocks are encountered, this meeting will seek to identify the cause and solution.

The team may also call for a meeting outside of the regular weekly meetings for any of the following reasons: unexpected trouble during a sprint, reassignment of tasks, evaluation of documentation prior to deadlines, or any other reason that the team is in agreement on warranting an impromptu meeting.

2.3 AGILE WORKFLOW

As stated previously, the team will be employing the Scrum-based SAGE process. The SAGE process outlines an agile development workflow. As an agile process, research and development will be coordinated in sprints of two-weeks. Individual and team effort charts, a project burndown chart, and velocity charts will be made to illustrate the team's performance before, during, and after these sprints. To assist the team in constructing these charts, managing sprints, and maintaining the product backlog, the tool Jira will be used.

2.4 COMMUNICATION WITH CLIENT

The team will maintain frequent and appropriate contact with the client throughout the entirety of the project. Communication with the client will be both direct and indirect as described below.

2.4.1 Direct Communication

At the end of each sprint, the team will email the client with a status report stating what was accomplished in terms that the client will understand and be interested to learn. This includes by default any updates regarding research performed, reevaluated agreements or proposed adjustments to documentation, and functional advancements in code. It is left to the client's discretion to request information pertaining to intra-team management and SAGE-specific practices. In response to the status report email, the client may request or decline a meeting to discuss any points in further detail.

2.4.2 Indirect Communication

Indirect communication is maintained with the client by providing the client with *Read-only* access to the team's BitBucket repository containing all iterations of documentation, presentations, and code. This allows the tech-savvy client to have a live peek into the team's progress and workflow. Any concerns or questions the client may have from this indirect communication may be brought to the attention of the customer proxy.

3 SOFTWARE DEVELOPMENT AND TESTING

Throughout development the team will adhere to the test plan outlined below. Following such a test plan will ensure all of the client's requirements are respected and will seek to reduce the rate at which flaws or bugs are introduced into the project codebase.

3.1 BEHAVIOR DRIVEN DEVELOPMENT

As per the SAGE specification, Behavior Driven Development (BDD) will guide the development process. BDD places emphasis on the customer's desired behavior of the software. This will be practiced through the use of stories that explain the needs and desires of an end user. All development then is to be guided by the desired behavior of the resulting code more than any other metric. After BDD has produced functioning code that behaves in the necessary manner, more rigorous testing may follow.

3.2 VERSION CONTROL AND PULL REQUESTS

All code is to be committed to the git repository hosted at BitBucket.org on a separate, feature-specific branch. When the feature is thought to be completed, a pull request must be made to seek approval before the branch is merged into the project master. It is upon this pull request that any testing must take place and be satisfied. A request that has been demonstrated to satisfy all existing tests as well as tests of any new functionality may be approved to merge with the project master. This workflow of branching and merging may be used recursively if a feature is divided into sub-features. In this situation, the sub-features will issue pull requests to be merged with the feature branch. When all sub-features have been merged, a pull request may be issued to merge with the project master.

3.3 TEST PLAN

Test	Test Description	Pass/Fail Criteria
Git Cloner Test	A given repository that meets requirements is passed to the “Git Cloner” component of the system.	The repository must be properly cloned to the local disk and accessible to the rest of the system.
Metadata Parser Test	A repository that exists on the local disk is passed to the “Metadata Parser” component of the system.	The parser extracts the relevant information (according to the dynamic metric) from the repository and produces a readable output to be used by other components in the system.
Database Test	Parsed metadata is passed to the “Database” component of the system.	Data is properly stored and is able to be retrieved for output without loss of information.
Hotspot Scoring Test	Metadata from a repository is provided to the “Hotspotter” component to be analyzed.	The hotspotter component produces an output stating the likelihood of bugs in detected files from the repository. The detection should match hand-calculated predictions.
Visual Display Test	All relevant metadata and an existing local repository are provided to the “Web Frontend” component.	The web frontend arranges the data (scoring and file structure) in one or more visible graphs that are accurate and understandable.
Client Acceptance Test	A beta, and later a finished, system is presented to the client.	The client confirms that all requirements are met and the system behaves as requested.