Project Specification

for the

Hotspotter Bug Prediction Software

CS 425 / CS 499 Senior Project

by

Nathan Reinhardt

Spencer Smith

Dylan Williams

of

Team HotSpotter

PROJECT-SPEC

Revision 1.2

As Of: 06 October 2015

Change Log:

Revision	**Change Note(s)**
1.0	• Initial release
1.1	• Standardized formatting per IEEE • Removed non-required design elements
1.2	• Added Signature section • Updated Specific Requirements and Verification

Reviewed and Approved by:

Name	Signature	Date
Nathan Reinhardt	_____	_____
Spencer Smith	_____	_____
Dylan Williams	_____	_____
Dr. Crk	_____	_____
Dr. Mayer	_____	_____

1. INTRODUCTION

1.1 BUSINESS AND DOMAIN DESCRIPTION

The client seeks a system that, when provided a *git* code repository, identifies files within the repository that have a high likelihood of containing bugs. Existing systems provide similar results using various static metrics (i.e. lines of code, cyclomatic complexity, etc.). The system to be developed will instead employ dynamic metrics that take into consideration the number of repository commits for a given file and the relative frequency of the commits.

The domain of the software is that of software developers, project managers, and open source contributors. Software developers currently use various bug tracking techniques and predict bugs via static metrics. Project managers must assign developers and testers according to hotspots detected within their codebase. Open source contributors can locate the most relevant portions of a codebase that is most in need of contributions by seeking hotspots.

1.2 PURPOSE

The system will be utilized by code developers and project managers to reasonably identify the files within their code-base that are in need of closer scrutiny by experienced developers. Using the system can also provide a better understanding of a particular repository to new developers. By identifying and flagging frequently modified code, new developers can see more immediately where their efforts may be most needed as files that have remained static for a considerable amount of time are likely to be either bug-free or insignificant.

1.3 CONCEPT OF PROPOSED SYSTEM / SCOPE

The “Hotspotter” system will predict the locations of bugs in a given code repository by identifying files that are being frequently modified in recent time. These “hotspots” will be given a score of the likelihood that they contain bugs.

1.4 PRODUCT OVERVIEW

1.4.1 Product Perspective

- *System Interfaces*
    - The web server will interface with the web page and database. The web server will be handle calculating data, sending data to web page and sending data to the database for storage. The web page will display the

data from the web server and send commands to the server. The database will handle storing and retrieving data for the web server.

- *User Interfaces*
  - The user interface will be a simple web page to interact with the remote database through the web server.
- *Hardware Interfaces*
  - Not applicable
- *Software Interfaces*
  - The system will use a software stack to implement the web frontend, server backend and database manager.
- *Communication Interfaces*
  - The web page will communicate through standard Secure HTTP since some of the information could be sensitive.
- *Memory*
  - The database will store every revision from the git repository so the amount of active repositories in the system will be limited.

- *Operations*
  - Normal user operations include:
  - Connect to repository
  - Calculate hotspots
  - Filter hotspots
  - Update repository
  - Special user operations include:
  - Specify update timeframe
  - Remove active repository from database

- *Site adaptation requirements*
  - Not applicable

1.4.2 Product Functions

The system will first connect a git repository specified by the user and download the said repository to a remote database. The system finds the hotspots in the repository with a user specified date range and dynamic metrics. The system displays the results in a visual and/or text format for the user to view. The results can be filtered by a file, function, developer or intensity of hotspot. The repository data is automatically pushed to the server by a given time frame or can be manually updated by the user. Multiple user can look at the results through different web interface instances.

1.4.3 User Characteristics

This software will be mainly used by software project team leaders and resource managers. The individual software coders may also find it useful for self-evaluation. The system won't require the user to have programming experience but will act as tool for them to understand where the troublesome code is located.

1.4.4 Limitations

The system will only be able to use git repositories for calculating hotspots. The accuracy of the scanning may depend on if the git repository followed a proper workflow with bug commits and fixes.

1.5 DEFINITIONS

Github – Distributed Repository
Bug – Unplanned problematic piece of code.
Repository – A place in which a collection of code lives.
Metric: a trackable code variable used to calculate metrics
Metrics: a method of measuring something, or the results obtained from this.
Hotspot: portion of code in a repository that is deemed a potential troublesome area.

1.6 STAKEHOLDERS

Dr. Igor Crk – Product Owner
Software Developers

2. REFERENCES AND OTHER STANDARDS

Semi-Agile Software Engineering (SAGE)

3. SPECIFIC REQUIRMENTS

3.1 EXTERNAL INTERFACES

GitHub – This will serve as our primary data source. The project will pull in repositories held within GitHub and run tests on it.

3.2 FUNCTIONS

Primary function will be to pull down a given GitHub repository. Run a sequence of tests and give a final score that will be used to find potential bugs within the project. The results will be displayed visually in a heat map of the repository. Each server API will handle their own error handling.

3.3 USABILITY REQUIREMENTS

The project will meet its usability requirements if it can be given a GitHub repository and find potential bug hotspots in the code with a measurable metric. The results will also be displayed visually and in a text format for storage.

3.4 PERFORMANCE REQUIREMENTS

Not applicable

3.5 LOGICAL DATABASE REQUIREMENTS

Not applicable.

3.6 DESIGN CONSTRAINTS

No design constraints have been issued by the client.

3.7 SOFTWARE SYSTEM ATTRIBUTES

The system will have documentation for all the backend APIs. The documentation will provide setup instructions to deploy the system on any server.

3.8 SUPPORTING INFORMATION

References:
http://google-engtools.blogspot.com/2011/12/bug-prediction-at-google.html
https://github.com/igrigorik/bugspots
http://landley.net/writing/git-bisect-howto.html
http://macbeth.cs.ucdavis.edu/fse2011.pdf

Problems Solved:
This software searches through git repositories and finds hotspots at the function level using dynamic metrics. The hotspots in the files are visualized with different shades of colors showing the intensity of the hotspot.

3.9 COURSE-SPECIFIC

Semi-Agile Software Engineering (SAGE)
Product Backlog – See Appendix A for details

4. VERIFICATION

We will develop a well-documented and modular code base to maintain extendibility. The proof-of-concept will show the server can retrieve a git repository, analyze the code base and provide a visualization of the results. The result will be tested against other code analyzing software product to prove efficiency. The client will have the final verification on the project that all the

5. APPENDICES

5.1 ASSUMPTIONS AND DEPENDENCIES

The GitHub repository uses git best practices
The Github repository uses git bug tracking practices

5.2 ACRONYMS AND ABBREVIATIONS

Repo : Repository
MEAN: Mongo, Express, AngularJS, NodeJS
TDD: Test Driven Development

APPENDIX A Product Backlog

Priority 1: Critical 5: Low	**Backlog Items**	**Estimated Remaining** (person-hours)
5	**Client Log-in** The client wants to log in to access account features.	8
5	**Create Account** The client can create an account for other users to view the hotspotting results.	12
2	**Add Repository** The client adds a new repository to the database via web interface.	16
3	**Manage Repository** The client can update/remove a repository from the database and change the automatic sync timeframe.	16
3	**System Sync Repository** The system automatically retrieves and updates the repository in database based off the current git repository.	20
1	**System Retrieve Repository** The system clones a copy of the git repository on server.	32
1	**System Analyze Repository** The system performs an analysis on the repository with specified settings and returns results.	56
1	**System Store Repository** The repository is translated into a format to be stored in a database for persistent storage	32
2	**View Hotspot Results** The client sees a visual representation of the results.	40
3	**Filter Hotspot Results** The client can filter the results based off different criteria.	24
5	**Save Result History** The system automatically saves a certain amount of results in the database.	12
5	**View Previous Results** The client can view previous result from a specific repository.	8
5	**Save Result Snapshot** The client can permanently save results to the database.	8
5	**Export Result** The client can download the text/visual results and save them locally.	12
	TOTAL:	296