

COMP SCI 4ZP6:
HubListener
Requirements Specifications Document

Authored By:
Ahmad, Zed
Jalan, Prakhar
Oliveira, Pedro
Selvathayabaran, Piranaven

Contents

1	Introduction	3
2	Project Drivers	4
2.1	The Purpose of the Project	4
2.2	The Client, The Customer, and Other Stakeholders	4
2.3	Users of the Product:	5
3	Project Constraints	6
3.1	Mandated Constraints	6
3.2	Naming Conventions and Definitions	7
3.3	Relevant Facts and Assumptions	8
4	Functional Requirements	9
4.1	The Scope of the Work	9
4.2	The Scope of the Service	9
4.3	Functional Requirements	9
5	Non-functional Requirements	11
5.1	Look and Feel Requirements	11
5.2	Usability and Humanity Requirements	11
5.3	Performance Requirements	12
5.4	Operational Requirements	12
5.5	Maintainability and Support Requirements	12
5.6	Security Requirements	13
5.7	Cultural and Political Requirements	13
5.8	Legal Requirements	13
6	Project Issues	14
6.1	Open Issues	14
6.2	Off-the-Shelf Solutions	14
6.3	New Problems	14
6.4	Tasks	14
6.5	Migration to the New Product	14
6.6	Risks	15
6.7	Costs	15
6.8	User Documentation and Training	16
7	Revision History	17
	References	17

1 Introduction

The following official document is the requirements specification document based on the Volere Template. It contains information regarding project drivers, constraints, functional/non-functional requirements and any other critical details that are defined under the templates definition. The specifications document is subject to change and any modifications will be noted in the Revision History Section.

2 Project Drivers

The following section of the SRS will explain the purpose of HubListener and its scope. It will also organize the direct and indirect stakeholders and how they will be using the service that is created from the project.

2.1 The Purpose of the Project

The User Business or Background of the Project Effort

Content: The user of this product would intend on evaluating their GitHub project using our tool, and look for any possible improvements to be made.

Motivation:

The key motivation for this project was our search for the answer to the following questions: What part of a GitHub project is the precursor to success? What key components do successful projects contain, that unsuccessful projects do not? How do we measure this ‘success’? At first, we assumed this measurement to be popularity on GitHub, or a thoroughly and well documented project. We are still in search of what specific metrics lead to success, and hope to discover a solution to this problem as we build and maintain the HubListener.

Considerations The user problem in this case is not serious, as most GitHub projects already use a variety of other tools to evaluate their growth and track their overall progress. However, by the end of the project, the user may clearly notice whether their project was a success or not. This is the moment where our tool can be valued, as it compares popular ‘completed’ projects to that of the users’. Any significant differences can be highlighted for the user, and thus, the ‘success’ problem can be solved. We would like to also give special considerations to the research done in empirical software, most notably the work done by Tim Johnston. [1]

Goals of the Project Content

The purpose of HubListener is provide users with relevant metrics, trends, and information regarding their GitHub project, and in some cases, push the user to make appropriate changes (can be organizational) that are intended to lean the project towards a more ‘successful’ one.

Measurement After analyzing a set amount of projects (e.g. 20), and pointing out any significant/valuable metrics for the user that lead to any changes (minor or major), we can safely say that we have succeeded with the project.

2.2 The Client, The Customer, and Other Stakeholders

The Client: Our primary client is Dr. Smith, who aims to use this software to measure the ‘quality’ of scientific computing software. Our secondary client would be Dr. Christopher Anand, who requires this for completion of the capstone course in exchange for credit towards the developer’s degree.

The Customer: The consumer of this service is any member of the open-source community. Being that the service is open-source, the consumer definition can

range but it is clear from our usability requirements that the service is aimed at adults who have expertise using open-source software and understand not to infringe on copyright.

Other Stakeholders: Other stakeholders include members of the open-source community who wish to fork the repository and improve/maintain it after the completion of the project. These stakeholders will be address on a case by case manner.

2.3 Users of the Product:

The Hands-On Users of the Product

Developers and creators with existing projects on GitHub are our primary user group for this product. Their subject matter experience can range from novice to expert, as we are simply analyzing their repository and comparing it against GitHub's 'best'. Our users can be developers, students, researchers, companies, and/or any organization looking to evaluate and improve their GitHub project in any way possible.

3 Project Constraints

This section describes the constraints on the HubListener project. Encompassed in these project constraints are the naming conventions, definitions, relevant facts and lastly, our assumptions.

3.1 Mandated Constraints

Below are the mandatory constraints that the HubListener tool is under :

Schedule Constraints: The service shall be available April 1, 2018. We want to launch the service by this date as it is the date that the project is due for assessment.

Budget Constraints: The project has a financial budget of zero dollars.

3.2 Naming Conventions and Definitions

A glossary containing the meanings of all names, acronyms, and abbreviations mentioned within the requirements specification. The following is a running, ongoing dictionary.

Naming Conventions List	
Naming Convention	Definition
GitHub	A web-based version-control and collaboration platform for software developers.
Node.js	An open source development platform for executing JavaScript code server-side.
HubListener	The service that is to be created given the details in this document.
Repository	A storage location that contains all of the project files (including documentation), and stores each file's revision history.
Command-Line Interface (CLI)	A text-based interface that is used to operate software and operating systems while allowing the user to respond to graphical prompts by typing single commands into the interface and receiving a reply in the same way
Cyclomatic Complexity	A software metric, used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code.
Node Package Manager (NPM)	Is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js. It consists of a CLI , also called npm, and an online database of public and paid-for private packages, called the npm registry.
Metric	A software metric is a standard of measure of a degree to which a software system or process possesses some property.

3.3 Relevant Facts and Assumptions

During the planning phase of this project, the HubListener Team outlined some key facts and assumptions that all stakeholders should know. These facts and assumptions will hopefully further solidify our requirements.

Facts

There are a few factors that influence the product. The first business rule to be addressed is the amount of GitHub repositories we can clone. As we are grabbing several projects for reference directly from GitHub, we are limited to clone at a ‘reasonable’ pace. Cloning the estimated 100 repositories in parallel is not something GitHub takes lightly, as it can be detected as abusive behavior by their automated measures.^[2] Another key factor that affects this product is the specific information we will be extracting from the initial list of projects. This information can vary drastically, and so we will be selective of the types of projects gathered. To elaborate, we may only take projects created in one specific programming language to even the playing field regarding any comparisons to be made.

Assumptions

There are a list of assumptions made right off the bat regarding this project:

1. The initial database of projects we will use for comparison, will be that of the most popular and ‘successful’ GitHub projects
2. The most popular projects on GitHub contain quality code and documentation
3. NPM packages used as dependencies are accurate, complete and correct.
4. Each user will have their own GitHub API Authentication token.

4 Functional Requirements

4.1 The Scope of the Work

Currently, there is no way to compare GitHub repositories against similar repositories within the ecosystem. There is no way to analyze your code against similar projects or see how your repository is trending in comparison to similar projects in the open-source community. The scope of the work is to find a way to solve this problem given the tools at our disposal.

4.2 The Scope of the Service

HubListener aims to provide a service which allows the user to compare his/her GitHub repository or any GitHub open-source repository against similar repositories in the ecosystem. The end user will be able to attain meaningful information such as metrics or trends that they can use to improve their current project and gauge how they are trending.

4.3 Functional Requirements

Below is a list of metrics that are to be outputted given a github project URL as input. This is followed by the functional requirements that we have identified for HubListener.

Functional Input And Output List	
Input	Output
GitHub Project URL	<ul style="list-style-type: none">- Cyclomatic Complexity- Cyclomatic Density- Lines of Code- Lines of Comments- Logical Lines of Code- Halstead Metric- Number of Methods- Number of Variables- Number of Issues- Number of Bugs- Number of Stars- Functional Coverage Score- Condition Coverage Score

Functional Requirements			
Reqt No.	Reqt Type	Description	Fit-Criteria
1	Functional	HubListener must take in a GitHub project link as input.	HubListener logs should validate that the input link is a GitHub checkout link.
2	Functional	HubListener must provide a set of metrics (refer to Functional Input and Output List above) to the end user, both printed onto the CLI as well as in JSON format.	HubListener logs shall verify the set of metrics outputted on the CLI and the JSON file.
3	Functional	HubListener must analyze the metrics and display one or more trends.	HubListener logs at least one trend.
4	Functional	Users must be able to customize which metrics to analyze	HubListener logs will validate at least one metric is selected.
5	Functional	User must be able to install, update or uninstall HubListener through the node package manager interface.	HubListener logs should validate a npm install, update and uninstall have succeeded when called upon.
6	Functional	HubListener must have a help dialogue accessible from the command line.	Hublistener logs should validate that the help dialogue exists.
7	Functional	Metrics gathered from the analysis of a repository must be added into a database	After HubListener has done an analysis, the results are viewable in the database. Logs validate an addition to the database.

5 Non-functional Requirements

The following sections go into detail about the non-functional requirements for HubListener and also the fit-criteria for each.

5.1 Look and Feel Requirements

1. The application should provide an easy and clear command line interface for user to use the service.
 - (a) **Fit-Criteria:** 80% of the users from our survey shall be able to navigate through the interface and utilize the service. The list of available options should be available on the main screen.
2. The application shall comply with Open standards.
 - (a) **Fit-Criteria:** 100% of users in our survey can verify that the application is easy to access/adopt and open for public review and debate.
3. Useful information (such as help, report issues, training) should be easily accessible.
 - (a) **Fit-Criteria:** 90% of users in our survey with knowledge of GitHub will be able to access areas for help, reporting issues and training.
4. When doing calculations or data handling like repository retrievals, the application should display an animated progress bar.
 - (a) **Fit-Criteria:** 100% of users in our survey can successfully verify that the application displays an animated progress bar

5.2 Usability and Humanity Requirements

1. The software must be simple for a person aged above 18 years, with knowledge of GitHub/open-source technology, in able condition to understand and use all its features.
 - (a) **Fit-Criteria:** 90 % of the users in our survey are able to use the application and all its features and deem the application understandable.
2. The application shall make it easy for the average user to find help guidelines.
 - (a) **Fit-Criteria:** 80 % of user in our survey are able to successfully find the help guidelines within one minute.

5.3 Performance Requirements

1. After the user provides their repository link, the application shall generate charts and metrics in a timely manner.
 - (a) **Fit-Criteria:** 80 % of the users in our survey agree that the charts and metrics were delivered in a timely manner.
2. The application shall save the users last request and results.
 - (a) **Fit-Criteria:** 100 % of users in our survey, after their first request, are able to navigate to the history section where they can view their last request and results
3. The application shall analyze the results and provide metrics and trends.
 - (a) **Fit-Criteria:** 100 % of the user in our survey are able to successfully identify at least one metric and one trend after running the application.

5.4 Operational Requirements

Operational Requirements are comprised of expected physical and technological environments.

Expected physical Environment

1. Users will use the application on their internet-connected computer
 - (a) **Fit-Criteria:** 100 % of the user in our survey are able to run the application on their internet-connected computer.

Expected Technological Environment

1. The application shall work on devices that have Node.js installed on their machine
 - (a) **Fit-Criteria:** 100% of users in our survey are able to run the application when they have Node.js installed.

5.5 Maintainability and Support Requirements

Maintainability

1. The software application is to be easily modifiable.
 - (a) **Fit-Criteria:** 80 % of the users in our survey are able to clone the repository and make fork requests.

Portability

1. The application shall be available on any computer operating system such as macOS, Windows and Linux
 - (a) **Fit-Criteria:** 100 % of users in our survey are able to run the application on their device irrespective of their operating system. The survey ensures that there is at least one user on all three operating systems defined.

5.6 Security Requirements

1. The application should not contain any authentication tokens hard coded within the source code.
 - (a) **Fit-Criteria:** 100% of the users in our survey were able to verify no authentication tokens in the source code.

5.7 Cultural and Political Requirements

1. The application should not display any offensive text or information
 - (a) **Fit-Criteria:** 100% of the users in our survey do not report any offensive text or information.
2. The application should be available in English.
 - (a) **Fit-Criteria:** 100% of the users in our survey are able to read the application main page and successfully determine that the language is English.

5.8 Legal Requirements

1. The application shall comply with the PIPEDA privacy act.
 - (a) **Fit-Criteria:** The application follows all [PIPEDA](#) rules as defined in the link.
2. The application shall comply with all relevant open-source laws.
 - (a) **Fit-Criteria:** The repository is Licensed under GPL-3.0 which abides by the regulations set by the [Free Software Foundation](#)

6 Project Issues

This section will contain information regarding how issues are being handled, some of the off-the-shelf solutions we will be employing as well as risks, costs and where to find documentation and training.

6.1 Open Issues

All open issues can be seen on our Issue tracking board. ([HubListener on GitHub](#))

6.2 Off-the-Shelf Solutions

Attempting to emulate similar application could greatly reduce the time needed to design and implement HubListener. If an off the shelf solution is available and fits our requirements, we will use it as part of the software, providing the necessary credit as needed. Below are the off-the-shelf solutions we are looking at right now:

1. [NodeDir](#)
2. [NodeGit](#)
3. [tmp](#)
4. [Simple GraphQL Client](#)
5. [Complexity Report](#)
6. [JSLint](#)

6.3 New Problems

N/A

6.4 Tasks

All open tasks can be seen on our Issue tracking board ([HubListener on GitHub](#))

6.5 Migration to the New Product

No new product to migrate to. This section is currently not applicable, but included for completeness.

6.6 Risks

All projects have risk. HubListener is no exception. Below are the risks identified and strategies as to how we will mitigating these risks:

Risk Table		
Risk No.	Risk Description	Mitigation Strategy
1	Inaccurate Metrics	As stated in the assumptions, we believe that the npm packages provided will be accurate. To ensure this, we will have test cases that verify the correctness of the npm package. Furthermore, any metrics coded by our team will be available to the open source community, such that if there are any problems in the calculations, issues on GitHub will be created and dealt with by the project team or the community.
2	Inadequate Metrics	HubListener strives to provide as many metrics as possible such that the stakeholders have as much information as possible. If there are any missing metrics which are required by the community, feature requests can be made on our GitHub. The requests will be resolved as per open-source policies.
3	Excessive Schedule Pressure	Although, HubListener is a time-constrained project, the HubListener team is governed by a supervisor and course instructor which will enforce that the project be completed on time. These quality gates ensure the milestones are delivered in a timely manner. Any changes in scope will be documented and relayed to the appropriate stakeholders.
4	Performance	HubListener performance is based on many variables. Benchmark and threshold testing results will be provided to the end User such that they can understand how this variation can occur.
5	Unproven Technologies	Although, the software is new in nature , it encompasses many proven technologies. Any changes or updates in the technologies will be relayed to the stakeholders.

6.7 Costs

There are no costs associated with the development of this project other than the time dedicated to development/documentation. It is developed under the open-source environment and therefore is usable for not-for profit purposes. In future, there may be a cost associated with maintaining the project, hosting the project or use by a professional company.

6.8 User Documentation and Training

User documentation will be available on the GitHub Wiki as well as on the npm description section.

Training will stem from this wiki and any additional information or changes made will be reflected in this document at a later point

7 Revision History

Table 1: Revision History

Date	Developer(s)	Change
November 1st, 2018	Piranaven Selva	Make Foundation for Specifications Document as per Issue #6
November 23rd, 2018	Piranaven Selva	Add Fit-Criteria to Functional/Non-functional Requirements as per Issue #30
November 24th, 2018	Piranaven Selva	Document: Constraints And Design Choice And Off-The-Shelf Solutions as per Issue #33
November 26th, 2018	Piranaven Selva	Maintenance

References

- [1] Tim Johnston. *Toolkit for Automatic Collection and Predictive Modelling of Software Metrics*. PhD thesis, 2016.
- [2] GitHub Inc. Limit on Cloning Repositories, 2017.