Requirements Document

Jake Ballinger

December 13, 2013

"A ship on the beach is a lighthouse to the sea."
- Dutch Proverb

1. Introduction to Document

1.1 Purpose of the Product

Many *Twitter* users have amassed a such plethora of tweets that it is cumbersome to manually search through them, an ability that many *Twitter* users would find useful. Presently, to do this, the user must download all of their tweets; however, because of the way *Twitter* rate limits its *API*, this can be difficult.

How wonderful would it be if we could search through all of our tweets? And what if we could do not only that, but what if we also could analyze the way we tweet?

This is the purpose of *Knightingale.*

This is the first release of *Knightingale.* This *SRS* will detail the requirements of the system; for more information, see the architecture and design documents.

1.2 Document Conventions

This document was written following *IEEE* conventions. It was formatted with LaTeX. This is the most high-level of the trio of documents; the architecture document is at a lower level of description, and the design document concerns itself with the implementation and algorithmic details.

Names, classes, and methods will be italicized, and both numbers and variables are modified with the $ wrapper in LaTeX.

1.3 Acronyms, Abbreviations, and Definitions

This document is intended for general consumption; a reader with little background in computer science or analysis should be able to reference this document for a simple understanding of the systems functioning. That said, the developers have referenced this document in both draft and completed form. As such, its lack of technical details has not invalidated its usefulness.

For the casual consumer looking to learn more about *Knightingale*, I suggest skipping to Section 1.4, which describes the scope of the product. From there, I would read Sections 2.1, the *Product Perspective*, which provides some of the background information and history of the product; 2.2, the *Product Functions*, which summarizes the major functions of the product, Section 2.4, which specifies the operating environment of *Knightingale*, and *Appendix A*, the glossary, which will be an aid to decode the documents terminology.

More technical consumers will enjoy reading the document in its entirety; however, for an expedited experience, I suggest following the same sequence as was outlined for the casual consumer, with some additions: Section 1.4, the entirety of Sections 2 and 4, and Sections 5.1 and 5.4, and *Appendix B*.

1.4 Scope of the Product

Knightingale is a system designed to inform any *Twitter* user of their tweeting habits. *Knightingale* could be used by social psychologists as a data-mining tool to obtain information about their clients or a population, or by companies to learn about a customer's interests to know which product to best advertise to them. For those looking to expand their presence on *Twitter*, *Knightingale* can be used to compare their tweeting habits to those of users with larger amounts of followers. We foresee *Knightingale* being used for both recreational and professional use.

1.5 References

1.6 Outline of the rest of the SRS

Section 2: General Description of the Product
Section 2.1: Context of Product
Section 2.2: Product Functions
Section 2.3: User Characteristics
Section 2.4: Operating Environment
Section 2.5: Design and Implementation Constraints
Section 2.6: Assumptions and Dependencies
Section 3: Specific Requirements
Section 3.1: External Interface Requirements
Section 3.1.1: User Interface
Section 3.1.2: Software Interface
Section 4: System Features
Section 4.1: Analytics System
Section 4.1.1: Description and Priority

2. General Description of the Product

2.1 Context of Product

Knightingale was developed as the final project for the *Computer Science 290: Principles of Software Development* class at *Allegheny College.* It is a new, self-contained product.

2.2 Product Functions

Twitter allows users to download a *.ZIP* file of all of their tweets. *Knightingale* reads in this file and stores the tweets in a database. From there, it can perform certain user-specified analyses:

- Composition of Tweets
- Content of Tweets
- User Interactions
- Search Functions

2.3 User Characteristics

The average *Twitter* user can use *Knightingale* to learn more about their tweeting habits. For this type of user, *Knightingale* is designed to be a fun tool.

Researchers and academics can use *Knightingale* as a tool to gather information. Psychologists can use *Knightingale* to identify patterns in a client's tweeting habits, and this can inform their research or therapy techniques. For example, if a psychologist is interested in studying self-esteem and social networks, they can define the ratio of tweets containing original content to those containing unoriginal content - quotations, retweets, or urls - as a measure of self-esteem. Without *Knightingale*, quantifying metrics like this would be unimaginably complicated.

2.4 Operating Environment

Development on *Knightingale* took place on both the *Mac OS X* and *Ubuntu* 12.04.

2.5 Design and Implementation Constraints

Knightingale is designed to run on *Ubuntu* 12.04; however, as was implemented in *Java*, it should be able to run on a variety of platforms. Hardware limitations are not an issue. *Knightingale* runs on devices without graphics

2.6 Assumptions and Dependencies

We assume that *Twitter's* method of downloading a user's tweets remains unchanged.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interface

The user interacts with the system through the command line.

3.1.2 Software Interface

Implemented in *Ubuntu* 12.04 *LTS*. *Knightingale* interfaces with *JCommander*, *Ant*, *JaCoCo*, *MAJOR*, *JDepend*, *JavaNCSS*, *SQLite* Version 3, and *Twitter4J.*

4. System Features

4.1 Analytics System

4.1.1 Description and Priority

The analytics system is at the heart of *Knightingale*; it is of the highest priority. Without an analytics system, there would be no *Knightingale.*

4.1.2 Stimulus/Response Sequences

Once the user has downloaded their tweets in a *.ZIP* file from *Twitter*, they can...[put this in]

4.1.3 Functional Requirements

Knightingale should throw an error message if the user is not reading in the proper file.

- REQ-1: Must allow user to connect to *Twitter* servers
- REQ-2: Must allow user to refresh tweet database
  - Includes tweets created since the download of the *.ZIP*
- REQ-3: Must allow user to specify a new *.ZIP* file
  - Reloading into the local database
  - Reinitialize the database by clearing existing tweets
- REQ-4: Offer at least 5 metrics about a user's tweets and tweeting behavior
  - a. Composition of Tweets
    - i. Number of retweets and replies
    - ii. Original content vs. non-original content
  - b. Contents of Tweets
    - i. Number of hashtags
    - ii. Most common hashtags
  - c. User Interactions
    - i. To whom does the user most commonly reply?
    - ii. Whom does the user most commonly retweet?
  - d. Search Functions
    - i. Find all tweets contained a certain word or phrase

5. Other Nonfunctional Requirements

5.1 Performance Requirements

There are no constraints on the speed, response time, or throughput of *Knightingale*, although it is understood that a faster system is ideal.

5.2 Security Requirements

Security requirements related to *Knightingale* are similar to those related to any password-protected information. It is the responsibility of the user to ensure that their password is secure if they do not want any information stolen.

Regardless, since *Knightingale* takes its data source from information in the public domain - a user's tweets - this concern may be unfounded.

The user will be authenticaed with *Twitter4j* and *Twitter's API* using *OAUTH.*

5.3 Software Quality Attributes

A master copy of *Knightingale* will be kept by the development team; it will also be stored on *BitBucket* in our version control repository. The developers will seek faults in the system and try to correct them. We estimate a high *MTTF* (mean time to failure).

Maintenance will be aimed at improving the quality of the system and including the planned features listed in *Appendix B*.

6. Other Requirements

6.1 Database Requirements

Knightingale uses a *SQLite* database to store all the information provided by *Twitter*. The database has two tables: one named `Tweets`, the other named `Users`. The `Tweets` table contains 10 columns and the `Users` table contains 2 columns. The `Tweets` tables is constructed in the same order as the information provided by *Twitter*, with matching column names. The `Users` table has *user_id* as its first column which gets populated with all replied and retweeted user, and *user_name* being the second column which is populated with a *Twitter4j* call to match user `IDs` to their *Twitter* profile name.

6.2 Deployment Requirements

Knightingale will be available on *Google Code* and from the *GitHub* version control repository.
Google Code: https://code.google.com/p/knightingale/
Github: https://code.google.com/p/knightingale/

6.3 Logistical Requirements

Knightingale must be implemented in *Java*. It must use *JCommander* to parse command-line arguments, *SQLite* Version 3 to store all tweets, store information about tweets, and extract information about the tweets, and use the *Twitter4J API* to extract current information from *Twitter*.

6.4 Testing Requirements

The entire system must be tested with *JUnit* test cases, and it must have a high level of coverage according to *JaCoCo*. Developers will regularly analyze the source code and test suites using *JDepend*, *JavaNCSS*, and *MAJOR*.

All analyses must be accessible through an *Ant* build system which supports compilation, documentation, and cleaning.

6.5 Additional Requirements

In order to interface with *Twitter*, *Knightingale* must be registered with the *Twitter Development Network*. Additionally, it must be able to access *Twitter* through the *Twitter4j* system.

7. Further Requirements

7.1 Additional Metrics

In the future, other metrics may be added to the system. Currently, the developers are considering adding a sentiment analysis metric to the system so that the user can determine whether most of their tweets are happy, sad, angry, neutral, annoyed, etc. If this is implemented, it will be added after the first release of the system. (This agent was independently developed on another team by one of the developers.)

7.2 Extra Credit Requirements

The developers can choose to implement alternative interfaces to the analytics system. For example, the development team may choose to implement a mobile application that allows user to upload *.ZIP* files and perform analyses. They could implement a web interface with a similar function, or a comprehensive *GUI* for *Knightingale*.

8. Appendix A: Glossary

API: Application Programmer Interface

GUI: Graphical User Interface. The *GUI* allows the user to interact with the program through images and mouse clicks rather than through the keyboard.

IEEE: Institute of Electrical and Electronics Engineers

LaTeX: A document preparation language. More information can be found here: http://www.latex-project.org/

MTTF: Mean Time to Failure

SRS: Software Requirements Specifications: This document is an *SRS*.

UI: User Interface

YAGNI: You Aren't Gonna Need It
.ZIP: An archive file format that supports lossless file compression.

9. Appendix B: To Be Determined List

This is a running list of future ideas that could be implemented. They were not implemented in the current project due to time constraints. As a development team, especially when creating the first release version of a systemm, we adhere to the principle of YAGNI.

1. Markov Model (for prediction)
2. Swing UI
3. GIS Analytics
4. Sentiment Analysis