Design Document

Jake Ballinger, Dibyo Mukherjee, Hawk Weisman, Gabe Kelly, Ian MacMillan

December 13, 2013

"...I have always found that plans are useless, but planning is indispensable."
- Dwight D. Eisenhower

1. Introduction to Document

1.1 Purpose of the Product

Knightingale is a *Twitter* analytics tool. Whether it's helping businesses better figure out to whom to market their products, aiding clinical psychologists understand the impact of social media on anxiety disorders, or making navigating *Twitter* easier for young adults, *Knightingale* is intended for all users.

The *Software Design Document* is intended principally for the development team and their professor at *Allegheny College*, *Dr. Gregory Kapfhammer*. This document provides a description of the functions of *Knightingale* at a low-level of detail.

1.2 Document Conventions

This document was written following *IEEE* conventions. It was formatted with LaTeX. This is the most low-level of the trio of documents; the architecture document is at a higher level of description, and the requirements document is more concerned with the high-level concepts and expectations of the procuct.

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

1.3 Scope of the Product

Knightingale is a *Twitter* analytics system. Refer to Section 1.4 of the Requirements Document.

1.4 References

1.5 Outline of the rest of the SRS

Section 2: System Overview
Section 3: Data Design
Section 3.1: Data Description
Section 3.2: Component Design
Section 4: Human Interface Design
Section 4.1: Overview of User Interface

2. System Overview

Knightingale was, at least in theory, conceived by *Dr. Gregory Kapfhammer* at *Allegheny College*. He assigned the creation of the system to the students in his *Computer Science* 290 *Principles of Software Development* class as their final project. Four teams of five students would each develop their own *Twitter* analytics system. *Knightingale* is one of those systems.

What sets *Knightingale* apart from its competition is its user-friendly interface, modularity, and expandability. The Knightingale command-line interface is exceptionally easy to use for any user with a basic background in command-line tools, and a `Swing` graphical user interface is planned as well. The structiure of the system makes it very easy to add new features and other components at any point in time.

To further develop an example given in Section 1.1, consider the situation of a clinical psychologist. It is common knowledge that a person's use of social media can be linked to their loneliness. The clinical psychologist might use the metrics provided by *Knightingale* to better understand her client. Another example concerns a person engaged in social-media marketing for their company on Twitter. Such an individual could easily make use of *Knightingale* to collect information about their company's Twitter feed to improve their marketing efforts or communicate information to their colleagues.

3. Data Design

3.1 Data Description

The information domain of *Knightingale* is the twitter archive.*ZIP* file downloaded from *Twitter*. The zip file is parsed as an ArrayList of Tweets using the ZipParser and TweetBuilder classes that is then stored in a SQLite3 database. *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.

Rows from the tweets table are usually extracted as ResultSets from the database and then converted to Tweets using the TweetBuilder class. ArrayLists of tweets are usually passed around in the various analysis methods.

The information domain of *Knightingale* is the *.ZIP* file downloaded from *Twitter*, as well as information pulled directly from *Twitter* through the *Twitter4j* API. When the user adds a new twitter archive to the *Knightingale* system, the Parser package is used to extract tweets from the archive file, which may then be discarded. The Database package is then used to insert these tweets into the SQLite 3 relational database. The database automatically generates a separate table of Twitter user IDs by selecting all of the unique user ID values that appear in the Tweets database. This table represents all of the Twitter users known by the system. These IDs are then associated with Twitter user names through communication with the *Twitter* API.

3.2 Component Design

The major components of the system are as follows:

Package `tweetanalyze` This package contains some fundamental blocks of the system:

`Tweet.java` : This class models a tweet as it is stored in the twitter archive zip file.

`TweetBuilder.java` : This class has methods to build tweets from `twitter4j Statuses`, `ResultSet`, and arrays of Strings.

`LogConfigurator.java` : This class sets up logging for the system.

Package `parser` `CSVParser.java` : Parse a tweets.csv file as found in the Twitter archive.

`ZipParser.java` : Extract the zip Twitter archive and calls CSVParser on the tweets.csv

Package `analytics` `Analyzer.java` : Master class for analysis that is extended by all other analysis classes.

`CompositionAnalyser.java` : Composition analysis for tweets

`CompositionAnalyser.java` : Composition analysis of replies and rewtweets.

`FrequencyAnalyser.java` : Analyses frequency of replies, and retweets.

`HashtagAnalyser.java` : Analysis of hashtags within tweets.

`SearchAnalyser.java` : Search through tweets using SQL `LIKE` operator.

`UserAnalyser.java` : User analysis i.e what users does the user interact with?

Package `database` `DatabaseHelper.java` : Access Layer for database. Has methods to create tables, insert tweets, an execute method for other classes to run queries among other methods.

Package `refresh` `AccessTokenHelper.java` : Get new OAuth access tokens and delete old ones from Twitter.

`TweetRefreshClient.java` : Use AccessToken to access TwitterAPI to get new tweets.

Package `ui`

4. Human Interface Design

4.1 Overview of User Interface

Knightingale's primary point of contact with the user is a command-line interface, although a Swing graphical user interface is a planned feature. The user may interact with the system using a series of command-line arguments to the `knightingale` command.

- `search <search term>`: search for all tweets with a word
- `refresh`: get new tweets for the user from Twitter's servers
- `add <path to archive>`: adds tweets from the archive with the specified path to the database
- `delete`: removes all user data from the database
- `frequency <type: replies | hashtags | retweets>`: performs frequency analysis on replied users, retweeted users, or hashtags, depending on the argument given
- `composition`: returns information about the percent composition of the user's tweets
- `cloud <type: replies | hashtags | retweets>`: generates cloud visualizations for the chosen global frequency

Additional visualizations, such as ASCII graphs for users in command-line only environments, bar charts and pie charts for displaying compositional data, and heatmaps for displaying geolocational data, are all planned features for *Knightingale*, and the modular architecture that the system uses for visualizations makes the addition of these and other visualizatyion types very easy.