5-2016

The Quality Attribute Design Strategy for a Social Network Data Analysis System

Ziyi Bai
zb6470@rit.edu

The Quality Attribute Design Strategy for a Social Network Data Analysis System

by

Ziyi Bai

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Software Engineering

Supervised by

Dr. Scott Hawker

Department of Software Engineering

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

May 2016

The thesis “The Quality Attribute Design Strategy for a Social Network Data Analysis System” by Ziyi Bai has been examined and approved by the following Examination Committee:

Dr. Scott Hawker
Associate Professor
Thesis Committee Chair

Dr. Christopher Homan
Associate Professor

Dr. Stephanie Ludi
Professor

Acknowledgments

Dr.Scott Hawker, I can confidently state that without your guidance I would not have accomplished my achievement. Your support has impacted my future for the better. Thank you for everything. Dr. Chris Homan, I am so thankful that you reached out to me and provided the requirements for this thesis. Your expertise in research, your guidance, and advice, have all been invaluable to me. Thank you so much. Dr. Mehdi Mirakhorli, without your help I would not have completed this thesis. Your expertise and great advice are so invaluable for this thesis. Thank you. Last but not least, I would like to thank all my friends, especially my roommate, for his encouragement and support.

Abstract

The Quality Attribute Design Strategy for a Social Network Data Analysis System

Ziyi Bai

Supervising Professor: Dr. Scott Hawker

An excellent software project can often identify quality attributes, expressed by the non-functional requirements. Quality attributes include modifiability, performance, availability, security, and so on. The nonfunctional requirements that address those quality attributes should be considered during the design phase. The quality attribute design approaches will be considered to achieve those requirements.

In this thesis, an architecture will be designed for a new social network data analysis system, is named Trowser, through several architectural approaches. Before starting the design phase, the source code of an existing social network analysis system, Gephi, is analyzed. The quality attribute design strategies are summarized from the analysis result and unite with the known design tactics to design the architecture of Trowser. The Architecture Tradeoff Analysis Method(ATAM) will be used for evaluating the quality of the architecture. Finding the good design strategies for social network data analysis system.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Social media has become an important part of communication in modern society. Social networking sites have become an important platform which people communicate. Among them, micro-blogging is growing rapidly. The microblogging phenomenon began in 2006, and now millions of users' microblogs generate massive content every day. Through microblogging people know the latest news, learn new knowledge, and share their lives. Twitter is a well-known implementation of micro-blogging that started in April 2006. Twitter messages, called tweets, have a maximum length of 140 characters. Relationships between people with a Twitter account are unidirectional, meaning that one user can follow another, but the user who is followed does not need to follow back that user. All tweets are public by default, and interesting tweets can be retweeted so that the original tweet can reach a wider audience. Through those data and content, the scientists can analyze human behavior or emotions [3]. In todays field of scientific research, software can help scientists effectively analyze massive amounts of data, greatly reducing time and labor costs. There are many data analysis tools available to help researchers analyze huge amounts of social network data. Some of the tools (Gephi, Graphviz, etc.) can visualize those data and help researchers to understand the relationships among them [4]. In this thesis, we want to design a new social network data analysis tool, named Trowser and based on Twitter data. The main goal of Trowser is to help researchers to handle geographic information and Twitter data.

Trowser is a web application to help navigate geo-socio-temporal activity on Twitter. The purpose of Trowser is merging social network, geospatial, and Twitter data. Trowser

can take a graph, where each node represents a collection of Twitter users. It draws each node at the mean location of all the geotagged tweets made by the members of each node. Trowser can filter out part of the graph based on graph features. There is a demo version of Trowser to show the basic features. However, the demo just reveals the functional requirements. The system requirements also include many non-functional requirements and non-functional requirements often influence the system architecture more than functional requirements do [22]. These requirements are qualifications of the functional requirements or of the overall product. Non-functional requirements express desired qualities of the application which will be developed.

Nonfunctional requirements, which also are called as quality requirements, refer both to noticeable qualities such as system performance, availability, and dependability, and also to internal potential problems, e.g., modifiability and security. Quality attributes are responsible for the quality of the system. The quality also supports the functionality of that system; the system will be hard to implement and use without a good quality. In this case, development teams need to achieve these requirements by using various specific techniques. We call these "Quality Attribute Techniques." Since the quality requirements often affect the system architecture more than the functional requirements do, most of the architecture design strategies chosen are based on the Quality Attribute Techniques.

In this thesis, we will design the architecture of Trowser follow by Trowser's requirements. These requirements not only include the functional requirements but also those nonfunctional requirements. In this case, we will think about the architecture techniques used for Trowser. Our research focus is designing a high quality of architecture for Trowser, so we need to analyze the nonfunctional requirements of Trowser. The nonfunctional requirements of Trowser will be responsible for the requirements for the quality attributes of Trowser. However, it is hard to test for an attribute if just provide an abstracted definition. Since that, we will create the quality attribute scenarios for those quality attributes which are required on Trowser. Followed by Trowser's nonfunctional requirements, the quality attributes for the architecture of Trowser mainly include modifiability, performance, and

security. To achieve those qualities, we will use architecture technologies.

For understanding which architecture technologies will be suitable for Trowser, we want to analyze the source code of the existing system, which have similar quality requirements. We try to find out the architecture techniques they used for the quality attributes and to analyze the roles of those strategies. However, we will not only follow on those source code to design the architecture of Trowser, since Trowser has its unique requirements. We will look for known architecture tactics and use them to design the architecture based on Trowser's quality attribute scenarios [3]. When we have a new architecture, we will use architecture evaluation techniques to evaluate the new architecture of Trowser.

Most development teams will move to software implementation after the design process is completed. However, it will be hard to maintain when people evaluate and test the system after the implementation. More and more teams are going to execute the evaluation before system implementation to reduce that effect. We will follow on the Architecture Tradeoff Analysis Method (ATAM) to evaluate Trowser's architecture. The final goal is to get a new architecture of Trowser, which maximizes those requirements, especially the nonfunctional ones. We hope this thesis can help people who want to design a software system which has similar nonfunctional requirements.

Chapter 2

Related Works

2.1 Trowser

This section will introduce the requirement of Trowser, and also the limitation of current version of Trowser.

2.1.1 Functional Requirement

Trowser can show two main types of data as shown in Fiqure 2.1. One is tweet data, which includes time, user name, tweet text, and the other is the location information and relationship between Twitter users, which are shown on the map. Each black dot is the centroid location of all geotagged tweets coming from the members of a maximal clique in the social graph (this representation is particularly useful for highly-interactive browsing of at-scale social network data). Red ovals represent spatial deviations from the centroid. When a user clicks on a node in this clique graph, only the tweets belonging to members of that clique are shown. The size of minimum clique is the number of users who follow each other, and the edge width between two cliques is the number of same Twitter users they share. The line is represented that if two different cliques have a specified number of Twitter users. The user can change the size of the clique and the width of the line. The list of tweets shows the messages of tweets that belong to the Twitter users in a clique. The user names are shown on the top left.

The demo version of Trowser is based on Meteor framework. Meteor is a platform that builds on Node.js and is used to develop real-time web applications [1]. Since Meteor is

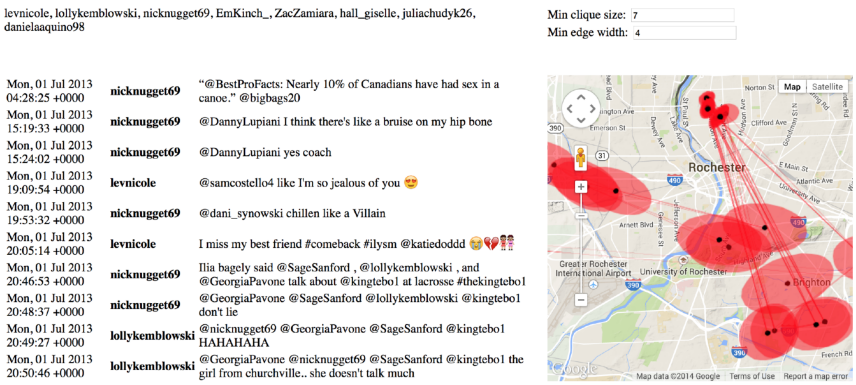

Figure 2.1: Demo version of Trowser

developed on Node.js, the client and server side use JavaScript as a development language. The demo version of Trowser has an architecture just based on the Meteor framework and Node.js without any other architecture tactics.

In my thesis, we want to design and implement these functional requirements in the following:

- Filter
- Query
- Visualize geographical information

Filter

The feature of the filter can provide filter the tweets by Twitter user name, date, the number of followers, and the number of replies. The user can enter specific elements to filter the tweets so that they can just check those specific information.

Query

Users are able to use database query language to find tweets through Trowser. As a requirement, Trowser's target users are familiar with database query language, and can use SQL to get the data. There is a window so that user can input the SQL, and Trowser will get this

SQL and get the data from database. The data which are queried will display on the list which is the same place that displays the tweets.

Visualize geographical information

The map in Trowser can show the geographical information base on the tweets which listed and user relationship. Through setting the number of minimum clique and then number of minimum edge, the different graph will be displayed on the map. User can click one red oval and the list of tweets shows the messages of tweets that belong to the Twitter users in that oval.

2.1.2 Nonfunctional Requirement

Follow on the purpose of Trowser, the researchers who want to analyze Twitter data always have new features to operate those data. Changes will always happen. Changes happen to add a new feature, to change or even retire old ones. Also, changes happen to fix defects, tighten security, or improve performance. The user who will use Trowser to analyze tweets data wants it is able to add new feature any time due to the scientists requirements. In this case, Trowser must be easy to change and be added the new feature, and there will be the lowest cost in the same time. The researchers, who want to put a new feature to Trowser, just need to change a few line of code to reach their goals. In this case, the new architecture of Trowser needs to have high modifiability.

Trowser also needs to handle mass Twitter data for querying tweets or visualizing the tweet location. The current version of Trowser uses MongoDB as the database, database store all tweets that posted in Rochester since 2014. The user will query tweets through Trowser, and the result also will merge with geolocation information which will post on the map. It is hard to operate those data without high performance. In this case, the customer brings the requirement that the new architecture of Trowser needs to have high performance.

Due to the feature of Meteor [1], the user can easily change the data through browser

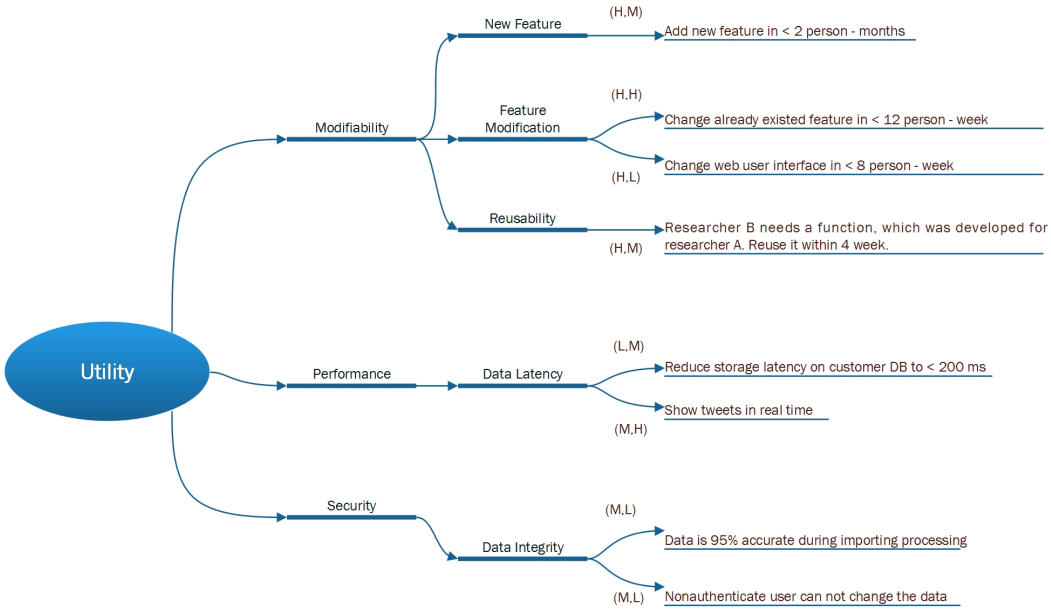

Figure 2.2: Quality Utility Tree

controller if the developer does not take care of security issues. And the current architecture cannot prevent any security issues. One of the goals for using data analysis tool is that reach the accurate data analysis results, and it is a difficulty that protects the data correct without high security. Based on those nonfunctional requirements, we can know that we need a new architecture for Trowser that needs to contain high modifiability, high performance, and high security.

To achieve those qualities, we create a quality utility tree which shown in Figure 2.2. The quality utility tree describes the quality attributes we can to achieve. Those quality attributes are described by quality attribute scenarios, those scenarios make the qualities testable. The quality utility tree also includes the importance of the scenario and the difficulty to implement the scenario. For example, (H, M) in the quality utility tree means that the importance of this scenario is high level and the difficult of implement this scenario is middle level. The quality utility tree also will be used to evaluate the architecture during the architecture evaluation process in our thesis.

2.2 Quality Attribute

2.2.1 Modifiability

The quality attribute of modifiability is about change, most of the cost of the typical software system occurs after it has been released [3]. Modifiability is one quality attribute of software architecture that relates to the cost of change and refers to the ease with which a software system can accommodate the change. Change will happen when to add a new feature or maintain the old feature. It is very hard to anticipate all of the changes, and most cost for software will be spent to evolve and maintain the system. All kinds of changes, which include anticipated and unanticipated, will spend the high cost when the system has been delivered. Another aspect, there are several platforms, particularly, mobile platform: many mobile applications have a multiple version for different platforms. In another word, software always keeps changing for the customers requirements. Development teams aim to reduce the cost of these adaptations, by addressing modifiability during the development.

The definition of modifiability is that:

The modifiability of a software system is the ease with which it can be modified to changes in the environment, requirements or functional specification [5].

Through this definition, we realize there are three main aspects in architecture modifiability. For Trowser, the requirement change will be most likely; the user will add or change the feature based on their new requirements for operating tweets data. Adding a new feature will just need less than two person-months. The requirement change is the main research objective during we design the new architecture. The requirement change also includes the modification of existing features. Base on the requirements, changing already one existed feature need to spend less than twelve person-weeks and changing web user interface just need less than eight person-weeks. Also, it is able to reuse a function within four weeks.

To design a software architecture which includes modifiability, there are some tactics can achieve which shown in Figure 2.3 [3].

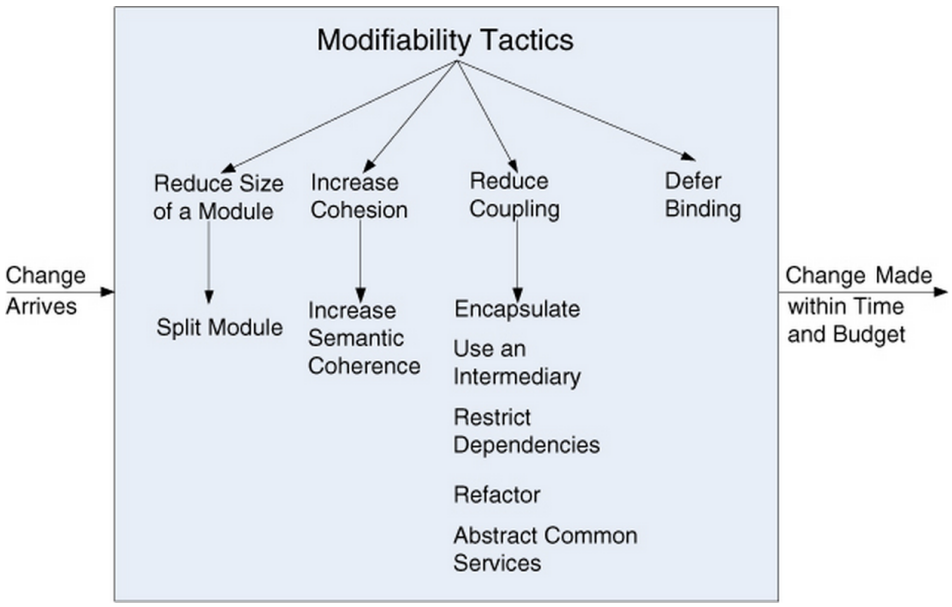

Figure 2.3: Modifiability Tactics[3]

2.2.2 Performance

An important issue for the engineering of complex software systems is determining overall system performance. Traditional software development methods focus on software correctness, introducing performance issues later in the development process. This style of developing has often been referred to as a fix-it-later approach. This approach does not take into account the fact that performance problems may require considerable changes in design, for example, at the software architecture level.

As a data analysis tool, Trowser need to process mass tweet data for analysis and visualization. For the performance of Trowser, data latency is the main problem we need to consider. In Trowser, it is able to reduce storage latency on the local database to less than 200 ms. Another part of data latency, the tweets are able to display in real time in Trowser. These two part of performance scenarios are performance requirements of Trowser, and in this thesis, we need to use some performance tactics to handle those performance requirements.

The goal of performance tactics is to generate a response to an event arriving at the

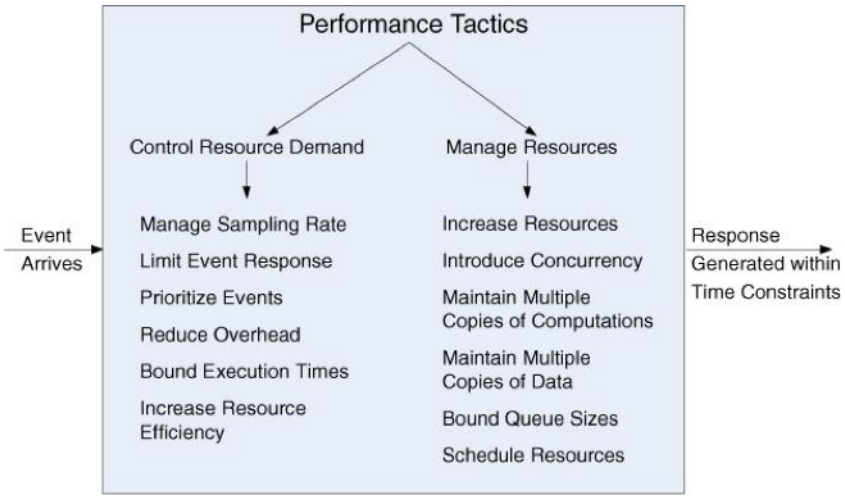

Figure 2.4: Performance Tactics[3]

system within some time-based constraint. The event can be single or a stream and is the trigger to perform computation. There are two main types of performance tactics: control resource demand and manage resource, which are shown in Figure 2.4.

2.2.3 Security

In the face of an increasing number of business services on open networks and distributed platforms, security issues have become critical. Devanbu et al. stated this as follows [10]:

Security concerns must inform every phase of software development, from requirements engineering to design, implementation, testing, and deployment.

In the design phase of software development, we should design security functions to satisfy the security properties of assets identified in the requirement phase. Specifically, we can design such functions using access control, authentication, cryptography, electronic signatures, and logging components or service. The goal of security is that system is able to detects, resists, reacts, or recovers attacks [3], as shown in Figure 2.5.

The purpose of considering security is we want to ensure the data accuracy. Users import their data to Trowser, and any operations can not change the data context from the user interface.

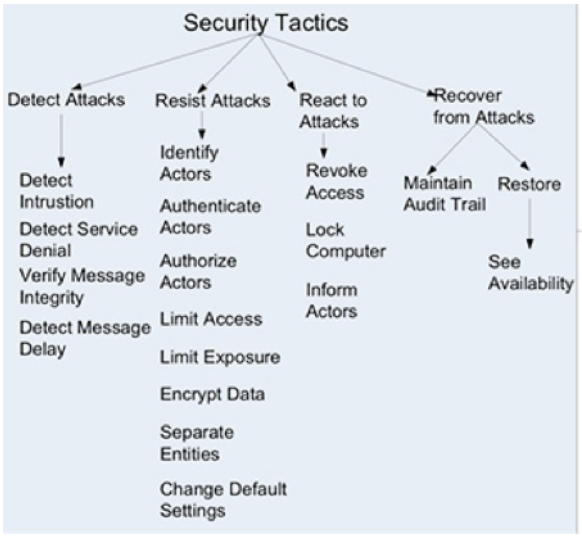

Figure 2.5: Security Tactics[3]

Chapter 3

Literature Review

3.1 Software Recovery

Understand a systems software architecture is the important facet of maintenance and research of software architecture. However, the maintenance and research of software architecture is challenged by careless, unintended addition, removal, and modification of architectural design decisions. To deal with that, software engineers are forced to recover a systems architecture from its implementation. We call them as software architecture recovery technology.

In this paper [14], the author compared six existing architecture recovery technologies: Algorithm for Comprehension-Driven Clustering (ACDC) [26], Architecture Recovery using Concerns (ARC) [15], Bunch [19], scaLable InforMation BOttleneck (LIMBO) [2], Weighted Combined Algorithm (WCA) [18], and a technique developed by Corazza et al. [8]. In this study, the research objects are nine variants of six architecture recovery techniques. The researchers used two different types of dependencies to compare them. There are several types of dependencies can be used as inputs in software architecture recovery technologies. Some technologies use control and data dependencies [11] [12] [25]; other techniques use static and dynamic dependency graphs [2]. In another study [16], the author pointed out that comparing with include dependencies, symbol dependencies are more accurate. They define that a symbol can be a function name or a global variable.

There are several more powerful tools with better GUI and more useable. In this study [28], the researchers extracted systems class graph through Sturcture101. Structure101 is a

kind of tool that extracts the systems architecture from source code. Structure101 have very great usability; there is a leveled structure map (LSM) that shows the system structure. The user can click the elements in LSM; it will show the sub-level elements which belong to that element. The types of elements are very rich. All of package, class, interface, method and value can be shown in LSM, and there are dependencies connect with elements. Clicking one dependency, the detail information will be shown.

Understand by SciTools can scan Ada, Cobol, C/C++, C#, Fortran, Objective-C, Java, Jovial, Pascal, PL/M, Python, and others. From this study [17], we know that using Understand to get several code analysis metrics, for example, Source Lind of Code (SLOC), Number of Children (NOC), and Response for a Class (RFC).

3.2 Architecture Approaches

3.2.1 Design Pattern

Design patterns are meant to solve recurring problems in software system design and so to improve the quality of software system [3]. Object-oriented design patterns can provide modifiability. The book [13] analyzed several design patterns and shown the scenarios how to use them. In a case study [23], the authors created a variable pattern which made by reusable feature types. The reusable feature types are belongs to several search techniques. They used three open source system to evaluate the quality, and the result shows that a customized pattern can effect high-quality detection results positively.

3.2.2 Performance

For service-oriented web system, there is a design approach which called interface-based design [7]. Service-oriented architecture is a way of designing a software system to provide services to either end-user applications or other services through published and discoverable interfaces [7]. Service-oriented architecture can provide modifiability for the system. However, this deign strategy will decrease system performance.

In the study [24], the authors talked about software performance engineering methodology. The software performance engineering methodology is the first comprehensive approach to the performance evaluation during the software development life-cycle. There are two models used in software performance engineering methodology, which are the software execution model and the system execution model. The software execution model can represent the software execution behavior, and the system execution model can represent the system platform, which includes hardware and software components. The result of the software performance engineering methodology can understand the whole software system and hardware system comprehensively.

There are several approach based on the software performance engineering methodology. Williams [27] and Smith [24] evaluate the performance of software architecture through the software performance engineering methodology. In their study, the software architecture is specified by using the Unified Modeling Language (UML) diagrams. In addition, the target model of the software architecture is used in the construction and analysis of the software execution model. Petriu et al. [21] propose several approaches which follow the software performance engineering methodology. Also, these approaches can help for building layered queueing network models of software architecture based on combinations of the design patterns.

3.2.3 Security

Boyd et al. [6] propose a protection mechanism to protect the system from SQL injection attacks, which can attack and destroy data through a web front-end. The architecture which based on this protection mechanism have a proxy between database and web server; that means it will decrease the performance. While we consider those three qualities, we need to think about the tradeoff between security with another two quality attributes. Dalton et al. [9] propose that the powerful security technology needs to have four characteristics: robust, flexible, end-to-end, and fast. Among them, flexible means that the security technologies should adapt to cover evolving threats. In this point, security can connect

with modifiability. In addition, the last characteristic is fast, which means the technologies should have a small impact on application performance. This characteristic also has a relationship with performance. In this case, we know that the quality attributes independent with each other. There is not only tradeoff but also complementarity. During the analysis architecture technologies, we can not just analyze them separately; we also need to think about them together.

Chapter 4

Research Question and Methodology

4.1 Methodology

4.1.1 RQ1

For our purpose, we need to design and build an architecture for Trowser, which contain high modifiability, high performance and high security. The architecture technologies of those existing social network data analysis tools can inspire us. The software architects of those data analysis tools also face some problems they need to bring the solutions. In general, those software architectures include the architecture technique for modifiability, performance and security. We will choose four social network data analysis tools (Gephi, Graphviz, networkX, snap) as research objectives. Through architecture recovery technology, we can get the architectures of those tools, and find out that:

RQ1: For modifiability, performance and security, which architecture techniques used in Gephi?

The architecture techniques will contain design pattern, architecture tactics or another solution on architecture level.

For answering this research question, we need to get the architectures through software recovery technology. Using Understand and Structure101 to get the architectures of research objectives, and them we can do analysis which of architectures work for specific quality attribute.

Architecture Recovery Tool

Understand a system's software architecture is the important facet of maintenance and research of software architecture. However, the maintenance and research of software architecture design decisions. To deal with that, software engineers are forced to recover a system's architecture from its implementation. That is software architecture recovery technology.

Understand is an IDE (Integrated Development Environment) designed from the ground up to help to extract architecture. Through Understand, we can easy get the dependency between files, and view which file depend on that file, this file depends on which file, or butterfly-dependency(show both of them)

Structure 101 is the tool that is able to help user to understand the source code. We use it to view to dependency and organize the thousands of source files in our codebase into a visual and enforceable module hierarchy.

Research Objects

In the thesis, we use architecture recovery tools to extract the system architecture of social network visualization tool Gephi. Through understanding and analyzing those extracted architectures, we can understand what architecture technologies they use.

Gephi is an open-source software for visualizing and analyzing large networks graphs. Gephi uses a 3D render engine to display graphs in real-time and speed up the exploration. Users can use it to explore, analyze, spatiality, filer, cauterize, manipulate and export all type of graphs. From the repository of Gephi, we can know that it is developed by 95.0% Java.

4.1.2 RQ2

There are several architecture technologies we found from those data analysis tools. Since the requirements are different between Trowser and those data analysis, only using those technologies which we found out is not enough for Trowser's specific quality attribute

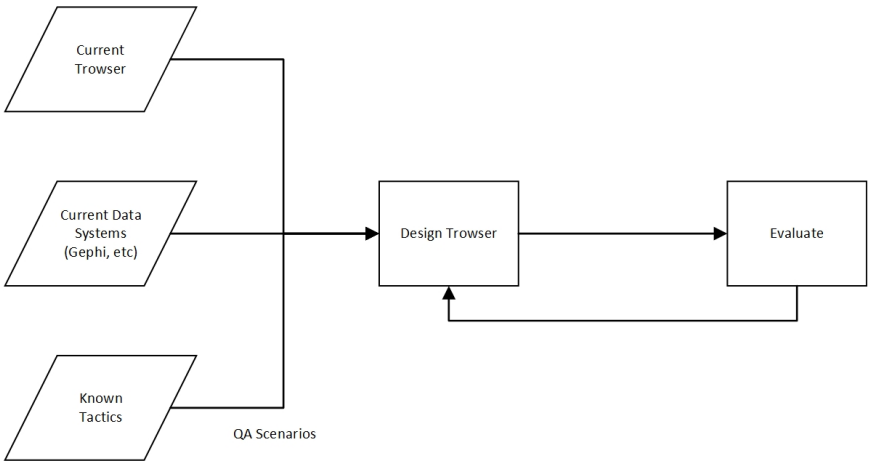

Figure 4.1: Work Flow of RQ2

scenarios. In this case, we need to use another known architecture tactics which can fit Trowser's requirements. Use those tactics which include what we found from research objects and known tactics to design Trowser architecture follow by quality attribute scenarios.

Our purpose is to design an architecture which can provide high modifiability, performance and security. For this purpose, we use the architecture technologies, which from research objects and what we know, to design the architecture of Trowser. In the same time, it will be the answer of our second research question.

RQ2: What architecture techniques can help to build Trowser with high modifiability, high performance and high security?

Figure 4.1 shows the work flow how to design the architecture of Trowser. We will analyze current architecture of Trowser, and combine the result after analyzing those research objects. Based on Trowser's quality attribute scenarios, we choose which known architecture technologies we will use into Trowser to design the architecture. After designing the new architecture of Trowser, the architecture evaluation will be used to evaluate the quality of Trowser. And we will improve the architecture based on the result of evaluation. After several iteration, we will get the architecture of Trowser which can achieve Trowser's requirements.

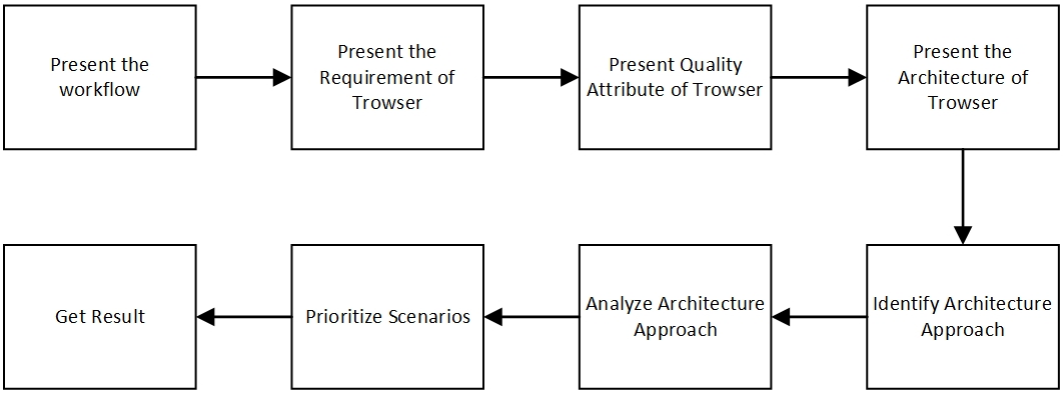

Figure 4.2: ATAM Work-flow

4.1.3 RQ3

We design the new architecture of Trowser to satisfy the quality requirements and will get the answer of the second question. However, there are three main quality attributes the development team need to satisfy in the nonfunctional requirements. These quality attributes have different priority based on the requirements. We need to evaluate or test the architecture to ensure that the new architecture has satisfied the requirements. Therefore, we have the following question which we want to find the result.

RQ3: Does the architecture we designed satisfy the Trowser's requirements?

We will evaluate the quality attributes after the architecture designed. The evaluation will follow the Architecture Tradeoff Analysis Method(ATAM), which shows on Figure 4.2. We choose the ATAM as the architecture evaluation since it not only evaluates an architecture satisfies particular quality goals but it also provides insight into how those quality goals interact with each other. Since Trowser has three quality goals with different priority, the ATAM is the good approach to evaluate how those quality attribute design strategies trade off against each other.

We will describe the evaluation method to the participants, and try to make them understand the work-flow. Trowser's design document will send to each participant before the ATAM meeting. That makes them have enough time to read the design document ,and

understand the requirements and architecture clearly. During presenting the requirements of Trowser, we will describe the features of Trowser to make the participants understand Trowser's functionality. It also help to understand the quality attributes of Trowser. We will focus on the nonfunctional requirements and then move to present the quality attributes of Trowser. We will show the quality attribute utility tree in this step. The quality attribute utility tree includes quality attribute scenarios, the priority of that scenarios, and the difficulty of implement. When the participants clearly understand the requirements and quality attributes of Trowser, we will describe the architecture to them. During this step, we will focus on how it addresses the quality attributes.

After the participants have a clear understanding of Trowser's architecture, we will identify the architecture approaches which are used in Trowser's architecture. We will describe each architecture approach and address to each quality attribute. The participants will analyze those approaches based upon the high-priority scenarios. The architectural risks, nonrisks, and tradeoff points are identified. However, a set of scenarios may be elicited from the entire group of participants. We need to prioritize those scenarios via a voting process, then analyze the architecture approaches again but use the highly ranked scenarios from the last step. Those scenarios are considered to be test cases to confirm the analysis performed thus far. We will collect all information during the ATAM evaluation and create a report as our result to present.

Chapter 5

Analysis

5.1 Architecture Analysis of Gephi

Gephi is an open-source for visualizing and analyzing large network graphs. Gephi uses a 3D render engine to display graphs in real-time and speed up the exploration. Users can use it to explore, analyze, specialize, filter, manipulate and export all types of graphs. From the repository of Gephi, we can know that it is developed by 95.0% java.

5.1.1 Modifiability

According to modifiability tactics, there are four main approaches, which are in the following, to provide the modifiabilirt.

1. Reduce Size of a Module
2. Increase Cohesion
3. Recude Coupling
4. Defer Binding

As a software for visualizing and analyzing large networks graphs, Gephi has the ability for query and filter data. In this case, filter and query data become the common services in Gephi. So, in the software architecture level, the designers reduced the coupling based on abstract common services. Throught *Understand*, we can get the structure of the filter part, which is in the following.

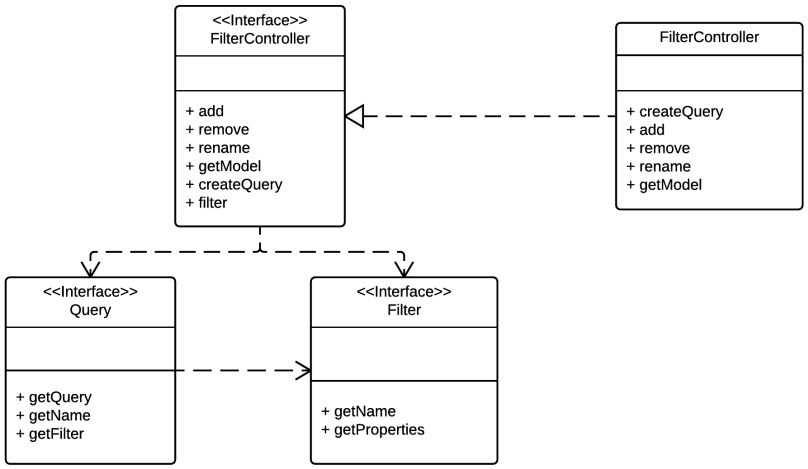

Figure 5.1: FilterController

Abstract Common Service

In this structure, there are several query functions in the FilterController.java. All of those query functions reference api.Query, which belongs to Query.java. Both of FilterController.java and Query.java belong to FilterAPI module. For implementation, there is a java file named FilterControllerImpl, which reference to FilterController. Meanwhile, FilterControllerImpl references to Query.

Separating the abstract and implementation part, it will reduce the coupling of architecture. Query, which is used for each filter action is abstracted as a common service. Abstracting common services may be cost-effective to implement the services just once in a more general form. Any modification to the service would then need to occur just in one place., reducing modification costs. From all of the modules' name, we can know that, most of functionality modules have one abstract module and one implementation module.

As a data analysis tool, Trowser would be able to query tweets base on several ways. In this case, abstract filer service as an interface, each query function will be implemented based on this interface.

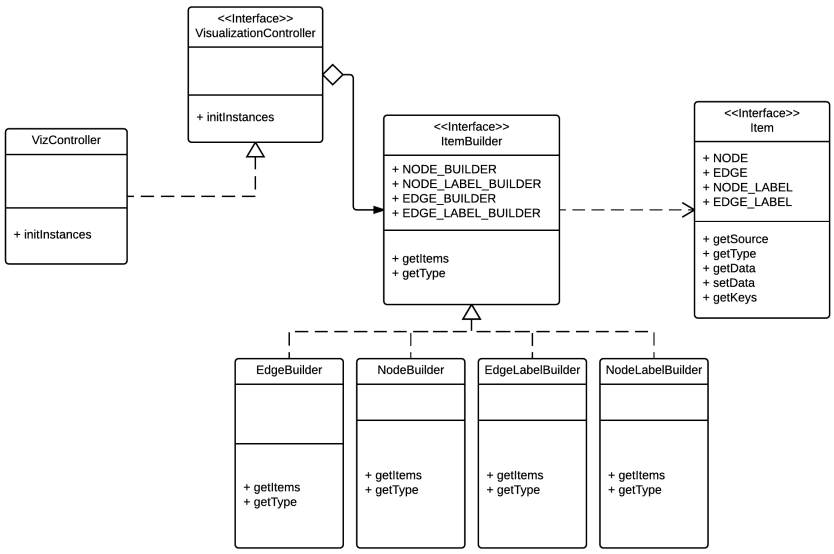

Figure 5.2: Builder

Builder

Visualization is the important functionality of Gephi, Gephi can display node and line for visualize the relationship and size of data set. It means that Gephi should be able to convert the data to many graphic formats. The problem is that the number of possible conversions is open-ended. So it should be easy to add a new conversion without modifying Gephi. To solve this problem and provide modifiability, the Builder pattern [13] is used in Gephi. As Figure 5.2 shows, there is an abstract interface named ItemBuilder, and there are several class which implement this interface in the builder package. In the builder package, there are several concrete builders, each of them created for single format.

The Builder pattern improves modularity by encapsulating the way a complex object is constructed and represented. Clint needn't know anything about the classes that define the product's internal structure; such classes don't appear in Builder's interface. Each concrete builder classes contains all the code to create and assemble a particular kind of product. The code is written once; then different Directors can reuse it to build product variants from same set of parts. Designers use Builder pattern to provide high modifiability which

attributed to abstract common services in modifiability tactics.

5.1.2 Performance

Since processing mass data and visualize them, the software need to provide high performance to avoid long waiting process data. As we mentioned in the previous section 2.4, there are several tactics can be used for performance. We can either reduce demand for resources or make the resources we have handle the demand more effectively. One of the ways to increase performance is to carefully manage the demand for resources. This can be done by reducing the number of events processed by enforcing a sampling rate, or by limiting the rate at which the system responds to events.

Limit Event Response

As a visualization tool, the graphs are the resource in Gephi. If there are too much data need to be visualized, there will be too many graphs displayed. This will be a problem if there is no performance tactic. We recovery the source code of Gephi, and check the structure and contents of the source code. We find that in the visualization model, there is a class named GraphLimits and several Graphics-related class import it.

Through limiting the number of graph, the software will bring high performance for users.

Bound Execution Time

Place a limit on how much execution time is used to respond to an event. For iterative, data-dependent algorithms, limiting the number of iterations is a method for bounding execution times. Gephi, as an application to visualize large network data, also have a mechanism to limit the time of long task.

On LongTaskExecutor.java, we found that there are some operating can interrupt or cancel those long task. LongTaskExcutor import Timer to implement canceltimer to calculate time for cancelable. Setting up a timer that how long the task can be cancelable and

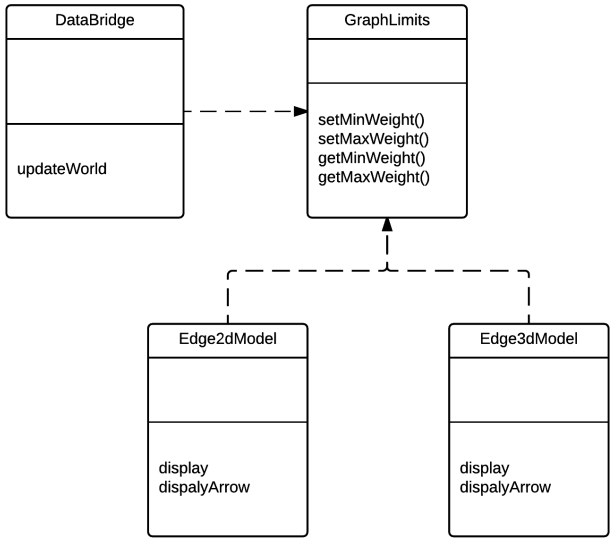

Figure 5.3: GraphLimits

then user can interrupt or cancel the task which has token a long time. In this case, this mechanism limit the task execution time through setting up timer.

5.1.3 Security

Security of software system is the system's ability of protect data and information from those unauthorized operators. Security includes three characteristics: confidentiality, integrity, and availability [3]. The confidentiality is the property that data need to be protected from unauthorized access. The integrity is the property that data is not subject to unauthorized operation. The availability is the property that the system will be available for legitimate use. Through these descriptions of security, we can clearly understand that the security's effect in the system. The general security requirement is that data should be protected and kept the accuracy during data operations. As the research object, data is very important to keep the accuracy. For protecting the data, the development teams need to design and actualize the security tactics. We analyze the source code of Gephi and Graphviz, and find the design strategy on those data analysis system.

There is DatabaseDrive module in the architecture of Gephi, which connect with the local database. We analyzed the code and find that DatabaseDrive can connect with different type of database, for example, MySQL, SQLite, and SQLServer. Those databases are local database, which store all of the data for analysis. Gephi is able to connect several database and it doesn't have network feature. In this case, user can operate the system without network. There are several benefits by using this tactic. Data analysis that can be operated without the Internet is able to protect the data and services from network attack.

Gephi can import data to local database, and process the data for visualization. We checked the code of Import module and want to find out what security tactics the development team used for Gephi. However, we just find out that there is a class neamed as LongTask. LongTask has a function to set a name for one import task, and there is another function to check the task name to make sure those imported data store to the same database. Checking long task name can ensure the accuracy during the system importing data to the local database.

5.2 Architecture Techniques for Trowser

After analyzing the architecture of Gephi, we found out what techniques are fit for social network data analysis. Abstract common service and using design pattern can improve modifiability and make reconstruction more easily. Limit event response and bound execution time are the two main approach to improve software performance. However, it is not enough for designing the architecture of Trowser, since there are many specific requirements only for Trowser users. In this case, we also need to use another tactics for those requirements.

The module view of the Trowser architecture (see Figure 5.4) shows the main modules in Trowser, there are three main subsystem which are TweetList, Visualization and DataIO. All of modules need to access database interface to get data, and there are several modules in each main modules. Query module works for query feature, which include getting tweets through SQL query. Filter module handles filtering tweets. Display model works

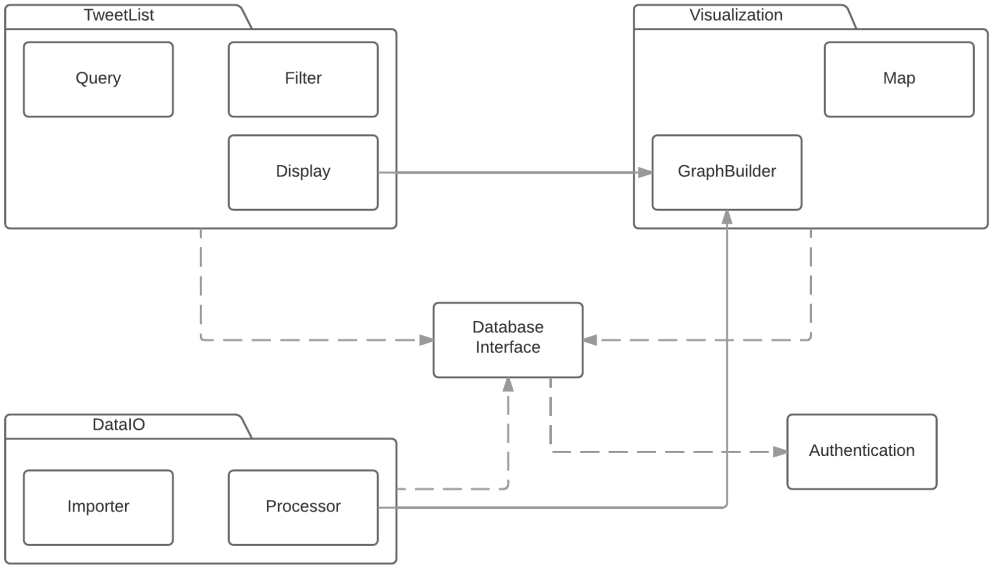

Figure 5.4: Module View

for displaying tweets on the list. Visualization model works for the whole visualization feature, there are two modules in the visualization module. Map will use three part map API to provide geographic information. GraphBuider module can get the tweet data from Display module to create graph on the map. DataIO works for data set, there are two modules in it. Importer module can read JSON file of tweets and import the data to database. Processor module can process data for visualization. Importer module also is able to access with Twitter API, so that Trowser can get the newest tweets.

The component-and-connector architecture view, which shows on Figure 5.5, provides a run-time view of the system. This allows the structure and behavior of processes such as data stores or authentication to be analyzed and improved. This view also provides an understanding of quality attributes such as performance and Modifiability. The component of Database Interface provides the flexibility for different type of database. It is able to connect with any type of database or multiple databases. There are three components connecting with Database Interface, which are able to access database to get the data.

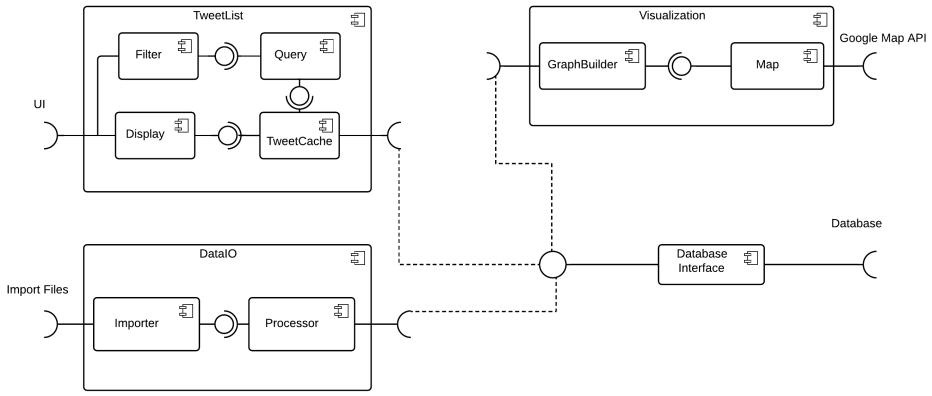

Figure 5.5: Components and Connectors View

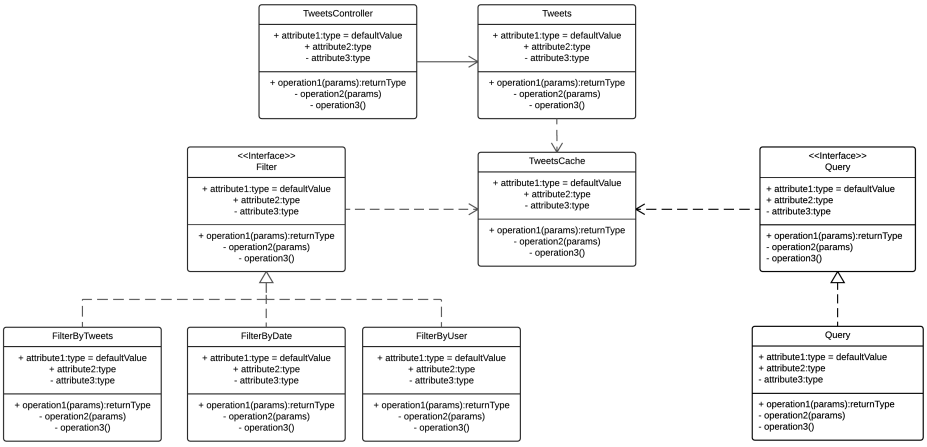

Figure 5.6: The class diagram of Filter and Query

TweetList Module

The purpose of TweetList module is to process tweets data and display. Follow Trowser's requirements, there are Query module and Filter module in TweetList module. Those modules work for the feature of querying tweets and filtering tweets. Fig 5.6 shows the class diagram of Filter and Query. Follow by the tactic of abstract common services, there are two interfaces for query and filter. Several entity classes implement the interface base on different functions, for example, FilterByTweets, FilterByData, and FilterByUser work for different filter based on different elements.

Using the tactic of abstract common services can provide flexibility for Trowser. For example, when developers want to add another filter which have a different filter element, they just need to create a class which implement Filter interface.

Visualization Module

Visualization module works for displaying the geographic information on the map. There are many graph will be displayed on the map. In this case, we use Builder pattern for building those different graphs. VisualizationBuilder is an abstract class, and different builder will implement VisualizationBuilder for different graph.

Using Builder pattern can provide modifiability for Trowser. When the developer needs more graph display, he just need to create a new builder for that graph through implementing VisualizationBuilder.

DataIO Module

The function of DataIO is controlling the database for obtaining the data from database. Simultaneously, DataIO will be responsible for preprocessing data for visualization. Figure 5.7 shows the structure of DataIO on class level.

DatabaseDriver class will connect with the local database, and it inherits Database interface. This strategy can provide modifiability, since it is possible that using a different kinds of database in the future. There are two processors in this module. Each processor class works for single metric graph (e.g., edge and cliques). These entity processor classes can process the data based on some special research requirements. Since the feature of processing data will be used in the most time, we create Processor interface following abstract common service. All of entity processor class should inherit Processor interface.

For Trowser can process huge volume data, we design TweetCache class follow by cache algorithm. When accessing large amounts of data is deemed too slow, cache is able to keep a small amount of the data in a more accessible location. Cache algorithm manage a cache of information stored on the computer in order to improve the performance.

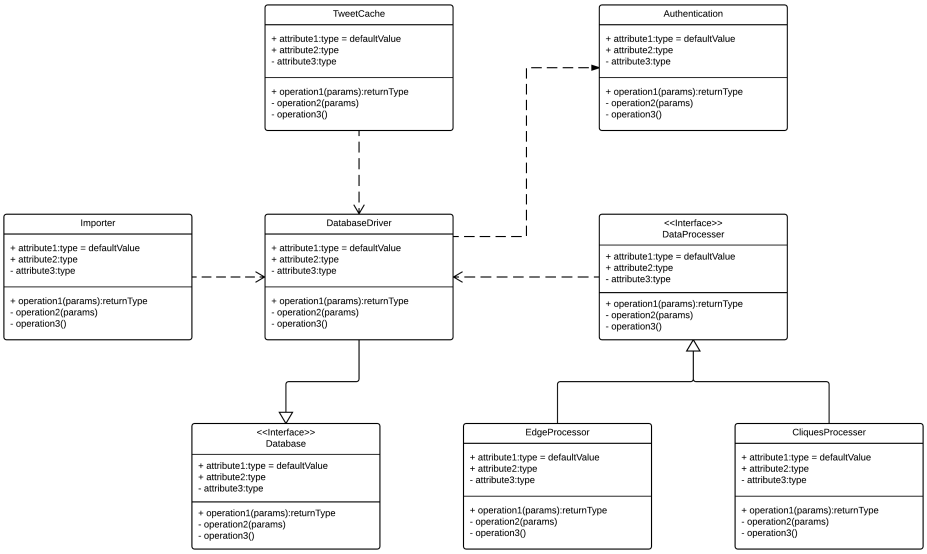

Figure 5.7: DataIO Module

TweetCache will follow by Least Recently Used (LRU), which discards the least recently used items first.

Database

User import the data to local database, and then Trowser will access to local database to process and visualize. Database is an interface which includes functions to connect with database. The class named DatabaseDriver is an entity class which inherits Database interface. DatabaseDriver will connect with local database. Different type of database will have different DatabaseDriver class. However, all of those class need to implement Database interface since connecting to database is a common service. This structure can provide modifiability for using different type of database in the future, and this tactic has used in Gephi.

Although we provide modifiability for using different local database based on the tactic of abstract common service, we still need to consider about performance. The type of database affects the performance of the system. We decide to use MongoDB as a default local database. MongoDB is a kind of NoSQL database. NoSQL database management

systems are useful during working with a huge quantity of data when the data's nature does not require a relational model [20]. MongoDB is a document database which belongs to NoSQL, and the document databases are designed to manage and store documents such as XML, JSON (Javascript Option Notation) or BSON (Binary JSON). The value column in document databases contains semi-structure data - specifically attribute name/value pairs.

5.3 Architecture Evaluation

We create a design document to explain the new architecture of Trowser. However, we do not ensure the new architecture we design based on several design strategies has satisfied the quality attribute requirements. We used the Architecture Tradeoff Analysis Method (ATAM)[3] to evaluate the new architecture we design, and the result will answer the third research question.

5.3.1 Architecture Tradeoff Analysis Method(ATAM)

We invited some person who have rich experience of designing software system architecture. The design document was send to each participant before the ATAM meeting. Therefor, the participants would have a little understanding for Trowser and can prepare the questions for the designer. During the ATAM meeting, the leader presented the meeting work-flow to make all of the participants have an overall understanding of this ATAM meeting.

Quality Attribute Scenarios

At "Present Quality Attribute of Trowser" step, the designer presented the quality attribute of Trowser based on the nonfunctional requirements, and shown the quality utility tree to the participants. There are eight scenarios in Trowser's quality utility tree, which show in the Table 5.1.

Table 5.1 shows the scenarios to describe the quality we want to achieve. The priority of each scenario shows in the table, which describes the importance and difficult of each

No.	Scenario	Category	Priority	Response Measure
No.1	A directive to add functionality. (Export selected tweets information from Trowser)	Modifiability	(H,M)	2 person- month
No.2	Modify Visualization Feature. Adding new graph and display on the map.	Modifiability	(H,H)	12 person-week
No.3	Changing user interface. Overlap ovals can separate when user click on them	Modifiability	(H,M)	8 person-week
No.4	Researcher B needs a function, which was developed for researcher A. Reuse it within 4 week.	Modifiability	(H, M)	4 week
No.5	Reduce storage latency on customer DB to les than 200 ms.	Performance	(L, M)	less than 200 ms
No.6	Show tweets less than 50 ms after database get the query	Performance	(M, H)	Average latency less than 50 ms
No.7	Data is 95% accurate during importing processing.	Security	(M, L)	Correct Data is Stored.
No.8	Non-authenticate user can not change the data.	Security	(M, L)	The data in database will be not changed by non-authenticate user.

Table 5.1: Quality Attribute Scenarios

scenario. For example, the priority of Scenario #1 is (H, M) in the table, that means the importance is high level and the difficult to achieve this scenario is middle level. The participants bring some questions for those scenarios. For Scenario #2 and Scenario #3 which address to feature modification, the response measures are too high. Based on the participants' experiences, it is hard to be defined as high modifiability if the system needs 12 person-weeks to change an already existed feature. The response measure of Scenario #4 is also too hard to achieve since each researcher will has his specific requirement for his own feature. To reuse other researcher's feature, there should be an architecture document to record the structure of that researcher's architecture. It is hard to understand and reuse

the code without understanding the architecture. Therefor, Scenario #4 which addresses to reusability only can be achieved in a few instances, that the developers will keep updating the architecture document.

Through presenting the quality attributes of Trowser, we found that there are three the scenarios which response measures are too easy to achieve. There are several reasons for those inaccurate measures. The designer's experience affects the accuracy of the response measures, which may be higher or lower than the requirements. For those response measures, we will analyze them again and let the customer review them. Based on new scenario response measures, we will modify the architecture of Trowser.

Architecture Approaches Analysis

We presented the new architecture of Trowser, and described the architecture approaches which are used for Trowser.For the modifiability, the participants agreed with using builder pattern on the visualization feature. Some participants thought there are another pattern which also are able to provide modifiability for visualization. However, using builder pattern is not a risky approach and can provide modifiability to achieve the modifiability requirement. For the feature of query, the original feature is able to write data to database. That is a high risk for the security, and it will affect modifiability since it too hard that abstract query service. The builder pattern provide the modifiability to create more types of graphs on the map. However, there isn't any architecture approach to solve the user interface changing. Furthermore, the frequency of changes in user interface is higher than the frequency of changes in the underlying logic. All of the design strategies didn't cover the modifiability of user interface, so the scenario of changing user interface can not be achieved.

We used cache to improve the performance of Trowser during query tweets from the local database. There is a risk that the size of cache affect the performance. If the cache size is too small, the quality of improvement will not obvious; if the cache size is too large, it will increase the hardware requirement. There is Database Interface in the new architecture

of Trowser, which provides the ability to connect with several different types of database. This design strategy supports the modifiability of database for Trowser. However, it is risky to maintain with different database technologies in the same time. Using single best database technology for Trowser will reduce that risk and provide performance in the same time. MongoDB is a good option for import and storing data from JSON files.

To achieve the security requirements, we designed that data need to be stored on the local database and use local-host to access. The participants thought storing data to the local database in user's own machine can protect the data from network attack. However, if a group of users who analyze the same data, how to handle this scenario is a problem that need to be considered. If the data stored on a server, it is a risk when user access the server on the outside. The easiest approach to solve this problem is restricting access address and authentication.

Final Report

Through the Architecture Tradeoff Analysis Method, we analyzed the quality attribute scenarios of Trowser. We found that there are several scenarios' response measures that are too high to achieve. Those measures should be reviewed with the stakeholders and make those measures more close to the quality attribute requirements. Overall, current design of Trowser's architecture can provide modifiability, performance, and security. However, there are some risk come form those architecture approaches. The tactics for different quality attribute also affect each other. Table 5.2 shows the outline of ATAM result.

5.3.2 Cache Performance

We create a cache demo to test our cache algorithm for performance. The purpose of this testing is want to evaluate the effect of cache size on performance. We used a same set of queries to query data from MongoDB, and the only difference is query data without using cache algorithm or with using different size of cache. Figure 5.8 shows eight execution time which include the execution without cache, and the execution with different size of

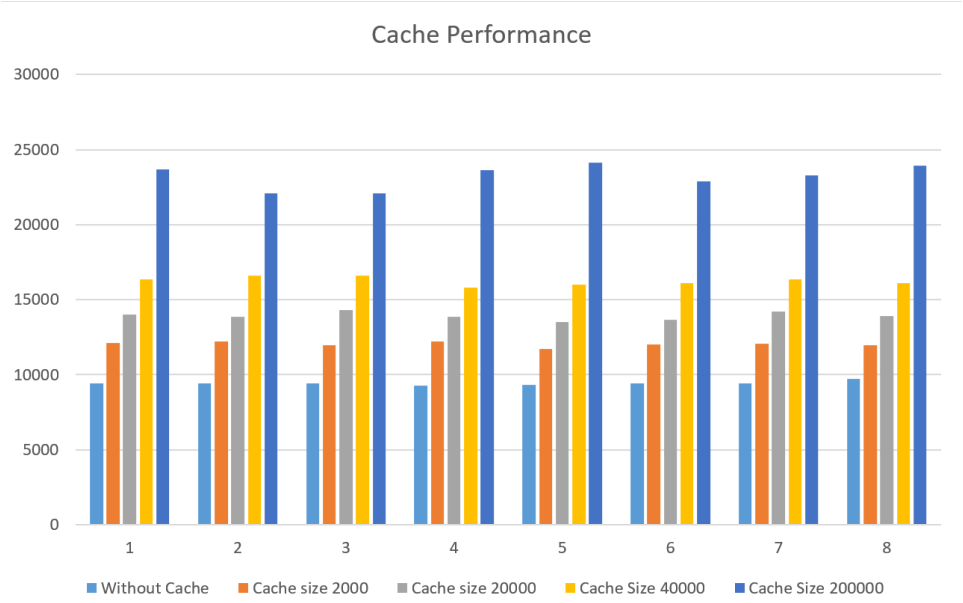

Figure 5.8: Cache Performance

cache.

The result in Figure 5.8 show that the execution time increase as the size of cache increase. The result shows that the size of cache affect the performance negatively. We think the reason is the maintain the space will decrease the query performance. In this case, we will remove the cache algorithm from our design strategies for performance.

Outline of Results Presentation	**Explanation**
ATAM Steps	A viewgraph listing eight steps of ATAM
The Requirement of Trowser	The description in section 2.1
Utility Tree	Figure 2.2
Scenarios	Table 5.1
Scenarios Discussions	The response measures of Scenario #2 and #3 are too easy to achieve, and should be reviewed. The detail summary in section of ”Quality Attribute Scenarios”.
Architectural Approaches	The summary in section of ”Architecture Approach Analysis”.
Risk, Nonrisks, Tradeoff Points	The size if cache will affect the performance. The feature of Query can write data to the database will affect the security and modifiability. Using Builder pattern doesn’t have a risk. Providing database interface for different DB technologies makes the system hard to maintain.
Other Issues	For a group of users, need to limit access address for security.

Table 5.2: Outline of ATAM Results

Chapter 6

Conclusions

Through analyzing the source code of Gephi and applying architecture tactics described in the literature, a new architecture of Trowser is designed by using several design tactics based on the functional requirements and the nonfunctional requirements. The quality of the new architecture is evaluated by the Architecture Tradeoff Analysis Method. Analyses of Gephi reveal that abstract common service can provide modifiability for the system, Builder pattern is used for creating different type of graphs. When there are too many graphs need to be visualized, Trowser will limit the number of graphs to display. Trowser listen the time of execution, and interrupt or cancel those long executions for system performance. We found Gephi support several difference type of database interfaces to connect different kinds of database. We didn't find another design tactics for security, the security may be have a low priority in Gephi's requirements.

We summary the design tactics used in Gephi through the source code analysis, and use some of them in the new architecture of Trowser. In addition, we use the architecture approaches which from known tactics. All of the approaches we used comply with the requirements of Trowser, especially the nonfunctional requirements.For example, Builder pattern is used for visualization and provide the modifiability, using cache algorithm to improve the performance, and using local database and authentication for security. The new architecture of Trowser is designed based on those approaches for achieving the business goal. The Architecture Tradeoff Analysis Method is used to evaluate the quality of the new architecture. We find that there are several scenarios' response measures that are too easy to achieve. The report of ATAM is created as the evaluation result, which can express the

quality of the new architecture we designed.

6.1 Limitations

Several limitations are important to be noted and understood in this work.

- Analysis Resource: The source code of Gephi is analyzed to find out the architectural techniques. The techniques are find out by the keyword and module name. Through this analysis approach, we can not find all of the architectural techniques used in Gephi. If we have the architecture document, that will be more easy and comprehensive to analyze the architecture of Gephi.

- Requirements of Trowser: In this design work, we ignore the effect of inaccurate requirements. In formal software development life-cycle, the requirements should be evaluated after the requirement phase. When getting the accurate requirements, the design phase can be allowed. Some of the quality attribute scenarios are not specific,

- Designer's Experience: Designer's Experience has a direct impact on the quality of architecture. Judging the design approaches accurately will make the quality of architecture batter, so that we can get a better result.

- Programming Language and the Type of Webserver: We just considered the technologies on architecture level in this thesis. However, the kind of programming language will affect the quality of architecture, for example the performance. Additionally, the different webserver also affect the selection of design strategies.

6.2 Future Work

Significant amounts of future work are possible with the new architecture of Trowser and the result of ATAM.

- Review the Requirements of Trowser: Based on the result of ATAM, we need to review the requirements with the customer to improve the scenario descriptions and response measures.

- Improve the Architecture: When getting the improved requirements, the architecture need to be update. New architecture approach will be used following by the ATAM result and improved requirements.

- Architecture Evaluation and Implementation: The improved architecture should be evaluated and improved several time, and Trowser should be implemented for further testing quality. More detailed and specific test data will be obtained during the test phase so as to better demonstrate the quality.

Bibliography

[1] Documentation - meteor.

[2] Periklis Andritsos and Vassilios Tzerpos. Information-theoretic software clustering. *Software Engineering, IEEE Transactions on*, 31(2):150–165, 2005.

[3] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.

[4] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.

[5] PerOlof Bengtsson, Nico Lassing, Jan Bosch, and Hans van Vliet. Architecture-level modifiability analysis (alma). *Journal of Systems and Software*, 69(1):129–147, 2004.

[6] Stephen W Boyd and Angelos D Keromytis. Sqlrand: Preventing sql injection attacks. In *Applied Cryptography and Network Security*, pages 292–302. Springer, 2004.

[7] Alan Brown, Simon Johnston, and Kevin Kelly. Using service-oriented architecture and component-based development to build web service applications. *Rational Software Corporation*, page 6, 2002.

[8] Anna Corazza, Sergio Di Martino, Valerio Maggio, and Giuseppe Scanniello. Investigating the use of lexical information for software system clustering. In *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, pages 35–44. IEEE, 2011.

[9] Michael Dalton, Hari Kannan, and Christos Kozyrakis. Raksha: a flexible information flow architecture for software security. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 482–493. ACM, 2007.

[10] Premkumar T Devanbu and Stuart Stubblebine. Software engineering for security: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 227–239. ACM, 2000.

[11] Roberto Fiutem, Giulio Antoniol, Paolo Tonella, and Ettore Merlo. Art: an architectural reverse engineering environment. *Journal of Software Maintenance*, 11(5):339–364, 1999.

[12] Roberto Fiutem, Paolo Tonella, Giuliano Antoniol, and Ettore Merlo. A cliche-based environment to support architectural reverse engineering. In *Software Maintenance 1996, Proceedings., International Conference on*, pages 319–328. IEEE, 1996.

[13] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[14] Joshua Garcia, Igor Ivkovic, and Nenad Medvidovic. A comparative analysis of software architecture recovery techniques. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 486–496. IEEE, 2013.

[15] Joshua Garcia, Daniel Popescu, Chris Mattmann, Nenad Medvidovic, and Yuanfang Cai. Enhancing architectural recovery using concerns. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 552–555. IEEE Computer Society, 2011.

[16] Thibaud Lutellier, Devin Chollak, Joshua Garcia, Lin Tan, Derek Rayside, Nenad Medvidovic, and Robert Kroeger. Comparing software architecture recovery techniques using accurate dependencies. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 69–78. IEEE, 2015.

[17] Ravish Malhotra and Megha Khanna. Analyzing software change in open source projects using artificial immune system algorithms. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*, pages 2674–2680. IEEE, 2014.

[18] Onaiza Maqbool and Haroon A Babri. Hierarchical clustering for software architecture recovery. *Software Engineering, IEEE Transactions on*, 33(11):759–780, 2007.

[19] Brian S Mitchell and Spiros Mancoridis. On the automatic modularization of software systems using the bunch tool. *Software Engineering, IEEE Transactions on*, 32(3):193–208, 2006.

[20] ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013.

[21] Dorina C Petriu and Xin Wang. From uml descriptions of high-level software architectures to lqn performance models. In *Applications of Graph Transformations with Industrial Relevance*, pages 47–63. Springer, 1999.

[22] Klaus Pohl. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[23] Ghulam Rasool and Patrick Mäder. Flexible design pattern detection based on feature types. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 243–252. IEEE, 2011.

[24] Connie U Smith and Lloyd G Williams. Performance solutions: a practical guide to creating responsive, scalable software. 2001.

[25] Paolo Tonella, Roberto Fiutem, G Antonio, and Ettore Merlo. Augmenting pattern-based architectural recovery with flow analysis: Mosaic-a case study. In *Reverse*

Engineering, 1996., Proceedings of the Third Working Conference on, pages 198–207. IEEE, 1996.

[26] Vassilios Tzerpos and Richard C Holt. Acdc: An algorithm for comprehension-driven clustering. In *wcre*, page 258. IEEE, 2000.

[27] Lloyd G Williams and Connie U Smith. Performance evaluation of software architectures. In *Proceedings of the 1st international workshop on Software and performance*, pages 164–177. ACM, 1998.

[28] Qifeng Zhang, Dehong Qiu, Qubo Tian, and Lei Sun. Object-oriented software architecture recovery using a new hybrid clustering algorithm. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, volume 6, pages 2546–2550. IEEE, 2010.