PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Zhuonan Song

Entitled
MEASURING MASHUP SIMILARITY IN OPEN DATA INNOVATION CONTESTS

For the degree of Master of Science

Is approved by the final examining committee:

Sabine Brunswicker Chair	
John A. Springer	
Kathryne A. Newton	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Sabine Brunswicker

Approved by: Michael Dyrenfurth 06/06/2016
Head of the Departmental Graduate Program Date

MEASURING MASHUP SIMILARITY IN OPEN DATA INNOVATION CONTESTS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Zhuonan Song

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2016

Purdue University

West Lafayette

ACKNOWLEDGEMENTS

I have received supports, helps and directions from so many people, and it is hard for me to summarize all of them in a single paragraph. First, I would like to say thank you to Dr. Sabine Brunswicker for accepting me as her advisee and guide me during my two-year studying as a Master student at Purdue University. She opened the door of open digital innovation for me and lead me to explore some inspiring research areas on innovation and computational thinking. Dr. Brunswicker helped me to breakthrough the mindset in studying and researching and start to focusing on making practical contributions to the work I was working on. I would thank my committee members, Dr. John A. Springer and Dr. Kathryne A. Newton, for their help during my progressing on thesis and advises on the questions I met during the research.

Thanks to my friends in the RCODI research group, Albert Armisen, Allison C. Lamb, Chien-Yi Hsiang, Eonyoung Cho and Bjorn Jensen. They helped a lot on my work in the research team and encouraged me to overcome any challenge I met during my research process. It is also enjoyable and inspiring to work with them, both personally and professionally. Thanks to the department of Technology Leadership and Innovation, which provides well-designed program for international graduate students and helps them to smooth the study experience at Purdue University, as well as get used to the cultural difference in a different country.

Special thanks to my girlfriend, Yuning. Without the encouragement and supports from her, I could never process multiple challenging tasks all together and finally make it happen to defense my thesis. Her understanding, which touches my soul deeply, allows me to be the person I want to be and reach to the dream that I should follow.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Song, Zhuonan. M.S., Purdue University, August 2016. Mashup Similarity in the Context of Open Data Innovation Contest. Major Professor: Sabine Brunswicker.

Contests have become an important instrument for fostering the development of novel open data mash-ups, in short open data innovations. Literature calls for new methods for measuring the similarity of open data mash-ups in order to identify code cloning and creative re-use of components of applications. Theoretically grounded computationally methods for identifying the similarity of open data contests are lacking. This study explores the similarity measurement of data-based mashups in the context of an open data innovation contest. Three different dimensions of mashup similarity are defined: code similarity, functional feature similarity, and visualized feature similarity. The results from the contest, including the source code, the running project and the descriptive documents, are collected as the research data for this study. Data analysis is based on the design and development of computational approaches to measure technology and functional similarity. The findings of this study will be helpful in better understanding the similarity of solutions in an open data innovation contest. This study contributes to the theoretical and practical approaches for similarity measurement, especially in the field of mashup development.

CHAPTER 1. INTRODUCTION

This chapter introduces the phenomenon of open data innovation contests addressed in this study. Further, it articulates the scientific problem of measuring similarity of open data mashups and presents the research questions. The chapter concludes by presenting the assumptions of similarity measurement design, as well as the scope and importance of this thesis.

1.1 Background

The concept of open data has been of increasing significance and regarded a source of innovation in the 21st century. Following the principle of open source, the open data movement supports the thought that some of the data should be accessible to everybody in terms of usage and distribution without any cost, restriction on copyrights or control on patents (Auer et al., 2007). Thus, open data is beneficial because it breaks the boundary of information and enables the accessibility of valuable data to different individuals and groups around the world. It also provides the machine-readable information instead of complex text to support data processing in information systems. Open data has become an important source of digital innovations (Perkmann & Schildt, 2015). The data.gov website, for example, is an open data platform generated by the United States government that offers developers access to over 200,000 datasets, which they can use to create digital innovations, such as mobile apps or web applications, that address governmental or civic problems in the areas of health, energy, or safety. To spur open data innovations, both governments and firms make use of open innovation contests to turn open data into novel and useful applications (Shadbolt et al., 2012). The Google Lunar-X contest is an example of how an innovation contest works: this contest promises $20 million to the team that can achieve a successful soft-landing of a specific spacecraft on the Moon, move the spacecraft over 500 meters, and transmit images back to Earth. Most of the participants in this contest have backgrounds in engineering or physics, which are related to the potential required skillset for a spacecraft project. The organizer, Google, will provide essential resources, which include the available spacecraft techniques, operational APIs on spacecraft hardware and historical data from the spacecraft databases, for the individual or teams of participants.

In the context of open data, challenge.gov is a platform that was launched by the U.S. government to leverage the creativity and skills of developers to create novel digital software applications for web and mobile applications. In addition, local open data hackathons, a particular form of open data contests, have become an important form of open data contests to spur digital innovations. Most of the applications developed are based on open data and are typically in the web application hybrid style commonly called mashups, meaning they utilize data from multiple sources to create an end-user service in a single software interface. Mashups are faster and allow for easier integration compared with website data portals and they are good at processing multiple data sources by accessing open data API's frequently (Yu et al., 2008).

Existing literature points out that the design of the open data contests is pivotal for the outcome of the contest (Boudreau et al., 2011). Indeed, researchers have identified some initial empirical evidence that transparency about what existing best-practices of others' solutions in a contest are would help the developers to learn from others and improve their own solution (Brunswicker et al., 2016). This learning activity will trigger reuse, which means the participants duplicate some components from other solutions and then their projects become similar (Bildhauer et al., 2009). At the same time, in the field of open data contests, transparency is also one of the key elements used to encourage participants to provide innovative solutions (Henriques, 2007). The positive part for transparency is that it will encourage the transformation of information inside entities. The use of transparency in an innovation contest process also will increase the ability of participants to address the problems, search for and locate external resources, and create better solutions. So generally there are three kinds of reuse activities: the greenfield projects, which stands for no reuse and creating novel projects; the cloned projects where projects simply copy others', and the augmented projects where the projects will develop some novel ideas based on the reuse of others' code (Brunswicker et al., 2016).

However, reuse in a transparent context will cause problems in identifying unique solutions created by the participants and distinguishing the components of the solutions that are reused by the participants. When the participants decide to learn from or directly copy the basic development and ideas from others, it becomes more difficult to determine whether or not their developments are because they reuse some code but also simultaneously put forth their own efforts. Plagiarism detection among all the project solutions is also required in a transparent contest.

1.2 Problem Statement

Given the need for identifying the degree and type of re-use of components of a solution in an open data contest, researchers require reliable methodologies for conceptualizing and measuring the similarity between two or more open data mashups. Higher similarity between two projects implies deeper reuse of development and design. Measuring similarities between software is not a new field of research. In the field of code similarity measurement, there are multiple ways to find similarities among different enterprise software (Yamamoto et al., 2005). However, mashups have a multi-layered architecture with different levels and can be different in similarity measurement compared with large-scale softwares. Some researchers have the viewpoint that the mashup architecture can be separated into three different dimensions: technical function, feature and user experience, and user visualization (Rodriguez & Chinea, 1998). The technical functions are the fundamentals to make the mashup work; the functional features can provide the interfaces to end-users to meet their requirements; the visualized feature will also influence the usage of the mashup as well as the decision-making process of end-users (Edberg et al., 2012). Thus, the similarity for mashup projects can also be measured in these three dimensions.

As for the level of technical similarity, even though there are sufficient approaches for measuring the similarity of source codes in different ways, none of these approaches is developed to deal with the specific code similarity in the context of an open data contest (Cosma & Joy, 2012). Moreover, because there are three dimensions in the field of mashup architecture design, the definition of similarity among mashup projects should also cover the dimensions in addition to the code similarity. Functional features and visualized features will also have an effect on the results of mashup evaluation besides the source code itself (Cosma & Joy, 2012). More researches are needed to address the similarity measurements in different dimensions for mashups. This study will explore the similarity of mashup solutions from two of the three different levels, which are the technical source code, and functional features. The two levels of measurement in this study are supposed to be developed with practical computational methods and theoretical supports.

This study expands the knowledge of mashup similarity measurement. It provides researchers and organizers of software-related contests with practical approaches for measuring code similarity based on a meta-analysis on the literature about software code similarity. Specifically, it focuses on the area of functional feature similarity by building the basic framework and methodology for definition and measurement. This study contributes to the function similarity in the area of psychology and computer science since the existing literature is lacking in this area compared with the one of technology similarities. The study also suggests further research on the relationship among the different levels of dimensions of similarity.

1.3 Scope

Mashup application development is a software development activity focused on the combination of data, visualization and interaction features. The design of mashup is one of the primary objectives of today's open data contests. It is distinct from other software development activities as the open data mashups combine different functions into a single interaction page, which requires more frequent data interaction and more personalized functions (Yu et al., 2008). Similarity of mashups will be part of the research field of solution evaluation due to the reuse phenomenon in the transparent contest.

This study will focus on the code and functionality feature similarity among mashups in transparent open data contests. The mashup projects studied in this research are all developed in the JavaScript coding language, which is the major language for web-based application development. The mashups also will follow the best practice of JavaScript development provided by w3school. Restricted programming libraries are provided to be applied in these mashups. By making such a scope restriction, this study will be able to measure the similarities without the interference introduced by the use of multiple coding languages as well as other programming concerns such as different programming styles and programming library tokens.

1.4 Research Question

The major questions for this research are as follows:

Q1: How can similarity be conceptualized for mashups developed in transparent open data contests?

Q2: How can similarity be computationally measured for mashups developed in transparent open data contests?

To answer these research questions, this research adopts a design science approach using the following principles: 1) theoretically develop a framework for measuring the similarity of mashup designs. 2) develop a computational method for measuring similarity of mashups. 3) testing and empirically validating the computational method in a test case approach specifically designed for this research (Berndt & Watkins, 2004).

By developing the framework, it extends existing frameworks on mashup design. The development of the computational methodology will draw upon a structured review of established methods of code detection algorithms. Further, it will enhance the existing mashup similarity measurement by borrowing from feature detection approaches in the area of psychology and decision support system to learn about the feature similarity in an information abstraction perspective (Tversky, 1977).

As for the evaluation of the computational method, this study will ensure the internal validity by exploring the existing code similarity measurement approaches and pick the one that fits the mashups in the context of open data contest best. The approach for the measurement of functional feature similarity is developed on the computation-based theories. The external validity in this study is tested for the specific context of the transparent open data context with the requirement for open data technics, limited programming languages and external libraries.

1.5 Significance

Different types of software may require different techniques to measure their similarities. This study clarifies the definition of similarity of mashup applications in the context of transparent open data contests. There is little research focusing on the area of code similarity and evaluation. However, in the existing literature about software similarity, less attention has been paid to examining specific cases from the functionality feature and user visualization perspectives. The two-factor model of software offers a different perspective in the area of software evaluation, where researchers could better understand the similarity among different software in terms of features (Zhang & Dran, 2000). This study will use homology modeling for the similarities measurement of functionality features and visualization features, which will provide a new perspective on the evaluation of software, especially mashup applications (Rodriguez & Chinea, 1998).

Rather than only focusing on these two dimensions, this study also will include the software similarity judgment and evaluation from the technique perspective. The gap between theoretical and analytical approaches is that previous studies are mainly focused on the similarity of technology functions rather than the functionality features of the mashups. These multiple approaches will help the researchers in the evaluation process for feature and user experience to make a fair scoring scale. Based on the similarities, judges and organizers can assign the same scores to different solutions with high similarity features even if they look different in a coding competition. If the approaches for mashup similarity could be identified, the contest organizers can have practical methods to evaluate the solutions of participants and detect plagiarism. They also will be able to analyze particular behaviors of participants based on machine learning and analysis approach in this study.

From the theoretical perspective, this study will enrich the research on mashup similarity in the context of transparent open data contests. Based on the definition of multiple dimensions of software mashups, this study will deal with the mashup similarity problems in a computational perspective in the different dimensions, which will benefit from further research on the correlations among each dimension. Moreover, by looking into the particular behaviors of participants in the coding contest using information system diagrams, this study also will be helpful to the research on the behavior analysis from a psychology perspective (Nickerson et al., 2008). The insights in this study on the software similarity and evaluation will help the in design of software transparency and reuse. The analytical definition of the concept and operational measurement process in the area of software similarity will benefit the follow-on reuse in the innovation systems (Boudreau & Lakhani, 2014).

1.6 Assumptions

The following assumptions are identified for this study:

1) All the mashup projects collected for this study are developed in the required programming language, which is JavaScript in the selected contest.
2) The external public service to support this study will follow the description of its theoretical design of fingerprint approach (Schleimer et al., 2003).
3) The algorithms and interfaces are properly designed to extract, transform and process the source code data.
4) The online dictionaries for JavaScript and other packages will cover all the language statements in any scenario.

1.7 Limitations

The following limitations are identified for this study:

1) The study will focus on two of the three levels of mashup dimensions: code similarity and functional feature similarity.

2) The generalizability in this study is limited and only applies to the study with 104 mashup projects developed during Purdue GreenIronhack.
3) The validation of the computational approaches will be tested on the designed test cases for the mashups in open data innovation contest.
4) As for the external validity, this study will only focus on the computational approaches in similarity measurement for JavaScript programming language and limited external programming libraries.
5) This study has the scalability to the open data context with limited programming languages and libraries.

1.8 Definition of Key Terms

Innovation Contest – An innovation contest is an activity or process of the industrial process, product or business development. It will bring participants in a contest into the context of competing to solve a specific problem (Piller & Walcher, 2006).

Open Data – Some digital information should be available to everybody to access and use without any cost, restriction on copyright or control on patents (Auer, 2007).

Transparency – "Transparency is defined as the perceived quality of intentionally shared information from a sender." It is practiced in companies, organizations, administrations, and communities (Schnackenberg & Tomlinson, 2014).

Mashup Application – "Mashup is a web page, or web application, that uses content from more than one source to create a single new service displayed in a single graphical interface." (Yu et al., 2008).

Code Similarity – "Code similarity is a computer programming term for a sequence of source code that occurs more than once, either within a program or across different programs." (Chilowicz et al., 2009).

Functional feature similarity – Feature similarity, or functional feature similarity is a metric defined to measure the similar components between different programs in the perspective of end-user functions (Tversky, 1977).

Visualized Feature Similarity – "Two geometrical objects are called similar if they both have the same shape, or one has the same shape as the mirror image of the other." (Bowman et al., 2012).

CHAPTER 2. LITERATURE REVIEW

This chapter provides a summary of recent studies in the areas of software similarity evaluation in the context of open data innovation contests, providing both a basic understanding of the methods in the subject area as well as the new methods for code similarity and functional feature similarity measurement.

2.1 Innovation Contest and Transparency

The definition of an innovation contest, as defined in the first chapter of this thesis, is a competition where innovators make use of their background knowledge, talend and past experience to create solutions for a specific challenge (Piller & Walcher, 2006). Organizers of innovation contests could include individuals, groups or firms, according to previous research. Organizers will design the innovation contests around a certain topic or context. The expected outcomes of these contests may vary based on the aim of the activities. The solutions or end products could be as simple as a prototype, an idea or a textual description, or they could be as complex as creating a practical working project. The organizer will circumscribe the target group of participants by defining the topic and degree of elaboration (Buillinger, Neyer, Rass & Moeslein, 2010). The organizer has to deal with the target of the population and the satisfaction issues, where the participants are more willing to enter the contests with better development supporting system and evaluation process, to involve more participation. Motivation is another factor contributing to one's willingness to take part in an innovation contest. It generally could be affected by introducing incentives such as rewards (Ogawa & Piller, 2006). After the process of participants solving the problems, the organizer has to find a method to determine the ranking of the solutions, which is called the evaluation process. Different evaluation methods may lead to different results in determining the best solutions. The methods used to rank the solutions also will have an impact on the outcomes of the whole contest in the field of further influence and promotion effects.

The degree of transparency is important to an innovation contest. The ideas behind transparency are about the issues on information processing and information technology (Sampaio, 2010). In the context of an innovation contest, transparency means the accessibility of the internal resources and other information such as the scores or source code of others to the participants. Looking at software development transparency, for example, Meunier (2008) states that software development transparency is the condition where all the internal programming parts are available to the users and other programmers. It could be through well-defined documentation or a highly commented project with source code. By making the source code open to other programmers, modules and functions in the original project can be reused to build new features, which will increase efficiency of development. A high level of transparency in an innovation contest will encourage the participants to more deeply engage in the contest and learn from each other about not only the approach of developing a certain feature but also to create novel ideas during development.

2.2 Code Similarity Detection

Previous researches have reported that approximately 5% to 10% part of source code in the large computer programs is duplicated code (Lague & Proulx, 1997). Programmers will reuse others' code by brute-force copying fragments when their projects have similar requirements to the developed ones. Especially in the objected-oriented programming languages, the reuse of code is more common because the code is well structured and easy to be used in a different programming context (Becker, 1995). There are several reasons and benefits for reusing the code from other projects. First, it would be easier for developers to make a copy of a code fragment compared with building the basic logic and variables for the program (Ducasse, Rieger & Demeyer, 1999). Second, the code in an existing software project is more likely to be robust and well-tested. When in terms of time and efficiency, programmers are also tending to reuse code fragments from existing projects to maintain high performance of their own project process (Ducasse, Rieger & Demeyer, 1999). Moreover, in some system development processes, the reuse of code is an essential strategy for the developing team to develop in a well-structured work arrangement (Baxter & Yahin, 1998).

Scholars have identified at least three approaches for detecting code clone and measuring the similarity of software in different perspectives and situations. First, the code similarity can be measured based on lexical unit, which includes strings and words in the context of software source codes. It will not use any textual transformation method on the source code before measuring the similarity and the source code will be delivered to the similarity calculating program directly in most cases. In this kind of approach, hashing is widely used for presenting the lexical units in the source code. Ducasse (1999) used hash data structures to store the strings in the source code in lines. Followed with internal textual comparison, Ducasse got the percentage of similar source code lines and then calculate the overall similarity among different softwares. The hashing of strings could also be used in generating dot plots, which supports the visual comparison among different files. In addition to the string-based lexical units, researchers have also developed token-based structures to act as a supplementary to the comparison. After the normalization of tokens in strings, suffix trees could be generated for tokens per line, which supports the similar units searching between two textual structures (Baker, 1995). Thus, without knowing the overall structure of the source code, textual units are helpful for measuring similarity directly in an efficient way. However, the accuracy of this approach is not guaranteed and varies in different coding styles and architectures of source code.

Second, from a structural perspective, people can use abstracted content structures to measure the similarities among different softwares. Abstract syntax tree (AST) is commonly used for building abstracted tree or graph structures in this approach. An abstract syntax tree, or syntax tree, is a concept in computer science using a tree data structure to represent the abstracted syntax structure in the source code in a particular programming language. It will capture the essential structure of the source code in a tree form, while omitting the syntactic details at the meantime. The idea of abstraction was raised by Baker in 1995, when the area was focusing on the large maintenance systems. Baxter and Yahin (1998) implemented the structure of abstract syntax tree in a practical approach to detect the code duplications in regular programs regardless of the language for developing. The basic problem in the code duplication and similarity detection is that different arrangement of fragments may lead to the same outcomes, which means that by modifying the orders or variable names of original code, the new program will be able to perform the same function in a different coding format. In AST, each node of the tree will represent a construct component in the source code. All the syntaxes are abstracted regardless of the real definition of names and arrangements. With the main idea of maintain the original logic in the tree, AST will store the conditions and judgment bodies in paths and the variables in the leaf nodes. The order in the paths will also represent the logic flow in the source code. In this way, the study will be able to detect both the duplication and the similarity using a few parameters. In the research of Baxter and Yahin (1998), the similarity is defined as the matching of two sub trees in ASTs for different source code. The similarity between two abstract syntax trees is defined in the following formula:

$$\text{Similarity} = 2 * S / (2 * S + L + R)$$

where S represents the number of shared nodes in the two ASTs; L represents the number of different nodes in the left AST; R represents the number of different nodes in the right AST. An example of generating and structure of the AST is shown in Figure 2.1.

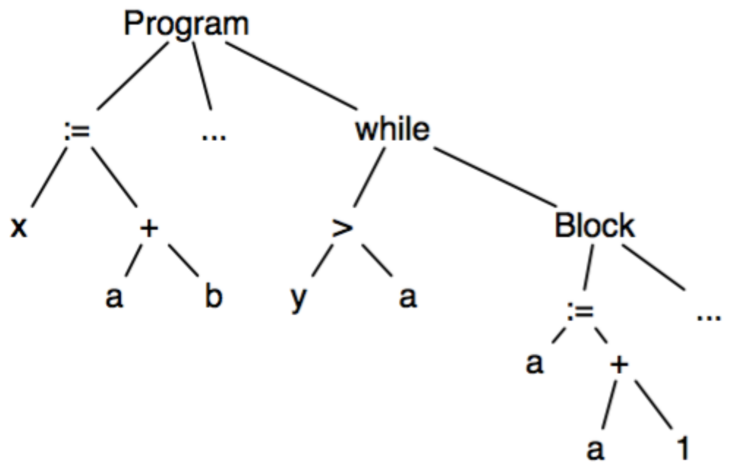
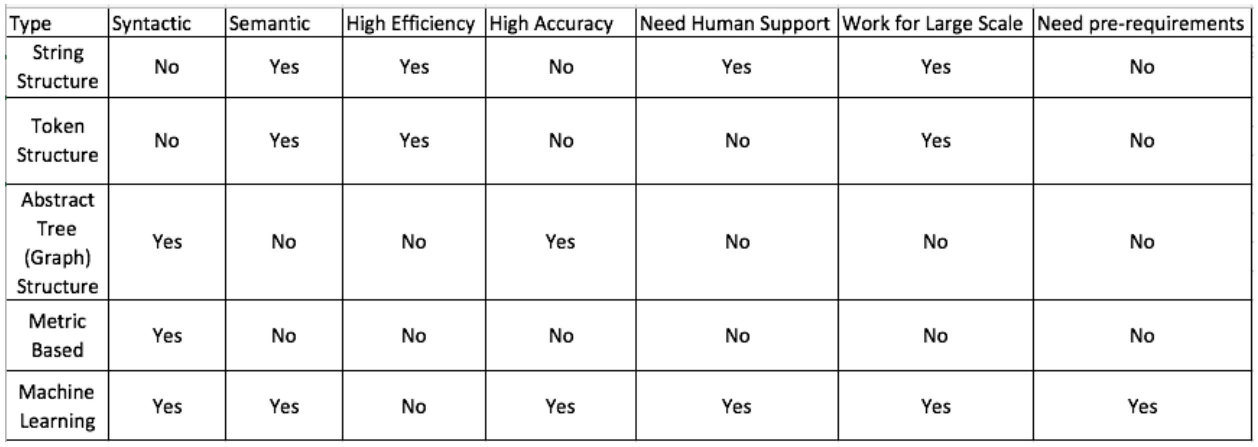


Figure 2.1 An example of generating AST

While the similarity of source codes in different softwares can also be measured in a metric-based approach. Researchers could cluster the vector of features to represent the procedures using neural net comparing metrics (Antoniol et al., 2002). Antoniol studied the code duplication detection in Linux kernel, which is a large, open source software system. By taking advantage of the neural network and the information distance in the source code, this kind of similarity measurement approach could easily be applied to different types of programming languages regardless of the difference in grammar and structure. Besides, it could also be used to locate specific code fragments during comparison, which will benefit the code revising process. On the other hand, the metric-based approach relies too much on the algorisms and is hard to be tested or verified all automatically. When it comes to website, the metric will detect begin-end blocks in the source code as basic fragments, which makes it hard to cover all the attributes stored in XML format.

In addition to these commonly-used approaches, the code similarity measurement could also be measured in the machine learning method. This approach will be able to deal with multiple types of similarities in different programming languages in high accuracy and efficiency (Basit & Jarzabek, 2009). The only significant prerequisite is to obtain enough existing results from successful similarity detection for the algorithm to

Table 2.1 Code Similarity Measurement Features



Type	Syntactic	Semantic	High Efficiency	High Accuracy	Need Human Support	Work for Large Scale	Need pre-requirements
String Structure	No	Yes	Yes	No	Yes	Yes	No
Token Structure	No	Yes	Yes	No	No	Yes	No
Abstract Tree (Graph) Structure	Yes	No	No	Yes	No	No	No
Metric Based	Yes	No	No	No	No	No	No
Machine Learning	Yes	Yes	No	Yes	Yes	Yes	Yes

The advantages and disadvantages of different approaches for calculating the code similarity could be summarized in the Table 2.1.

2.3 Functional Feature Similarity Detection

The feature similarity is based on the similarity of signals, views and interactions. Rothkopf (1957) tried to research on the signal similarity based on the 598 subjects recognizing Morse Code signal pairs. The pairs were presented in a randomized order and the result would be compared in the percentage of agreement of the subjects. The idea of this kind of similarity detection is kind quite old, but some of the major fields are built based on this the subjective approach. Law, Roto and Hassenzahl (2009) developed a survey-based approach for understanding, scoping and defining the user experience and product features, especially in the area of software. The main aim of their survey was for the promotion of active discussions in the area of user experience, which would be explored by a group of people who were active in the community. Statements, Definitions and Background were the three sections in the questionnaire used in the survey. With the result of this research, researchers would be able to learn about the definitions on a certain level of agreement and in different perspectives. They could also have reflections on the defined features to other experiences.

Tversky (1977) provided an approach to measure the feature similarity in a set theory perspective. In his research, features were resented in distinctive clusters. Each cluster was a subset of a group of features that were learned. The similarity between objects was presented as a measurement of their common and functional features. There would be overlapping among the different clusters in the research. Moreover, he tried to present the feature structures in a form of generated tree. There were no overlaps in different sub trees and each leaf was unique in the tree structure. The feature tree could be interpreted as a horizontal graph or hierarchical clustering scheme. The length of arc represented the weight of the cluster that was followed by a certain feature.

A process was developed by on Tversky (1977), Holzmann and Smith (2000) on feature verification of software. Their research was to develop features as properties which could be handled within a defined logic. There would be a lookup table for the logic requirement checking and system verification. The features were also divided into different subgroups without violation. Nikerson and Corter (2008) also developed a diagram approach for the feature detection in the information system design. The diagrams were used to represent the logical structure (Figure 2.3) and content of feature groups (Figure 2.2). They could show the result of spatial information in diagrams, which would be able to present both the logical connections and content of the features.

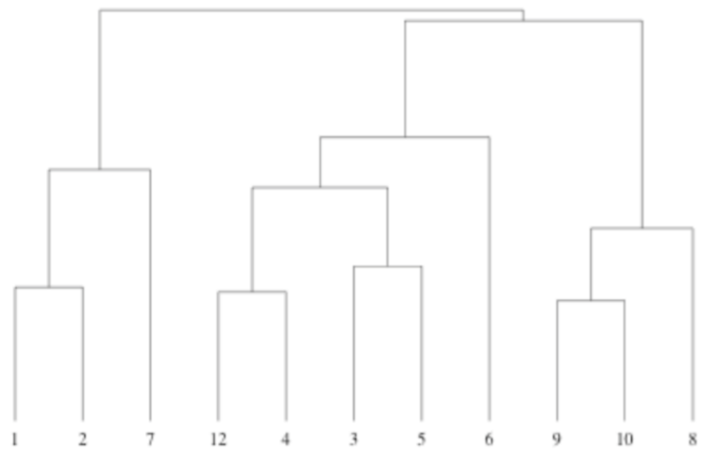


Figure 2.2 Clustering of the nodes (features) in a logical form (Nikerson & Corter, 2008)

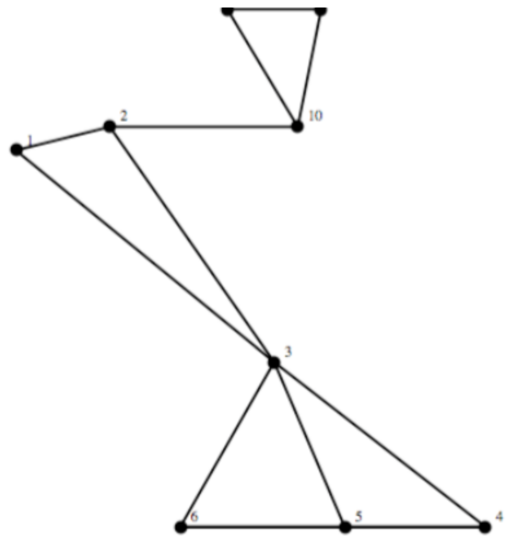


Figure 2.3 The logic structure of the features (Nikerson & Corter, 2008)

There is existing research on the measurement of functional feature similarity for softwares. However, most of them are based on human coding or some other subjective approaches, which would be inefficient for the measurement process (Branson et al., 2014). Feature of softwares is well defined in the field of computer science. Researchers also want to find computational approaches to measure the functional feature similarity and make use of the source code. Deep learning, for example, is a machine learning method which will be able to measure the function similarity based on source code (Weston et al., 2012). But it will not address the requirements and interactions from end-users. In order to do that, this study needs to learn from the psychological design and find the clusters of nodes for the end-user functions. In the theory of decision support system (DSS), functionality could also be defined based on the decision parameters provided in the system (Özacar, 2016). Özacar used structured data and parameters to define the function, which was designed to meet the requirement of the users on a data-related website tool. The variables were defined prior to the experiment and were filled with practical parameters in a subjective approach. The DSS relies largely on the case data and the example procedures, which means that the study can ran a scenario-based checking to list all the significant parameters before starting the real experiment (Sharda et al., 1988). The results showed that the functionality of decision support system could be well developed by defining the basic units of decision parameters properly, which can be applied to the computational matching of parameters for similarity measurement.

CHAPTER 3. FRAMEWORK AND METHODOLOGY

The research framework, sample set, testing methodology and the specific nature of data collection are introduced in this chapter. First, the purpose of this study and the detail methodology with the reasons of why those approaches are chosen for the research are described in this chapter. Second, this chapter contains the detail of the data collection process during the research followed by a plan of data analysis. At the end, this chapter presents the results of a pilot test to ensure the validity of the whole methodology and analysis plan.

3.1 Research Design Background and Settings

The project implements a design science approach to develop algorithms for mashup similarity measurement based on the project data collected from the open data contest at Purdue University. [1]There will be at least three sessions for this competition and this study collected data of project source codes from one of these sessions. Students are only allowed to participate in one of these sessions. An online community will be built for participants to communicate and interact with each other. There will be four phases in each session (Brunswicker et al., 2016). Participants will be required to submit their current iterations of the project at the end of each phase, which include the source

[1] This study is designed based on the NSF grant with the grant number 107673 sponsored by the Science of Science and Innovation Policy (SciSPI) program of SNF, which is developed by the research center of open digital innovation at Purdue University.

code, documentation and project package ready to be built. The goal of this study is to apply the practical computational approach for defining and measuring the similarity of mashup applications in the open data contest in the dimensions of technology, functionality feature and visualized feature. All the participants should be the students at Purdue University in West Lafayette.

A student sample size of approximate 30 students from a Purdue graduate class will be encouraged in our coding contest in the selected session. All students in these activities have the opportunity to participate in this study regardless of age range, gender, or ethnicity. During the coding contest, all the participants are required to develop a mashup application based on open data to solve local practical problems. The mashups should use JavaScript as their major programming languages and Google Map API is required to be enrolled during the development process. The mandatory requirements also include that participants should make use of at least one of the three JavaScript data visualization packages, which are D3.js, Arbor.js and Sigma.js. During each hacking phase, participants will get their feedback scores at the end of that phase and the running projects and source codes will be available to all the participants. They can learn from each other’s code and ideas as well as reusing some outstanding development segments in the source code.

3.2 Research Design and Framework

The original data collected from the research would be the source code of the projects, the running projects and the descriptive documents for the projects. It would be challenging to maintain these data before the analysis section. In this study, the three dimensions of data will be operated separately to get the processed data. There will be specific approaches for dealing with the data in each dimension.

3.2.1 Code Similarity

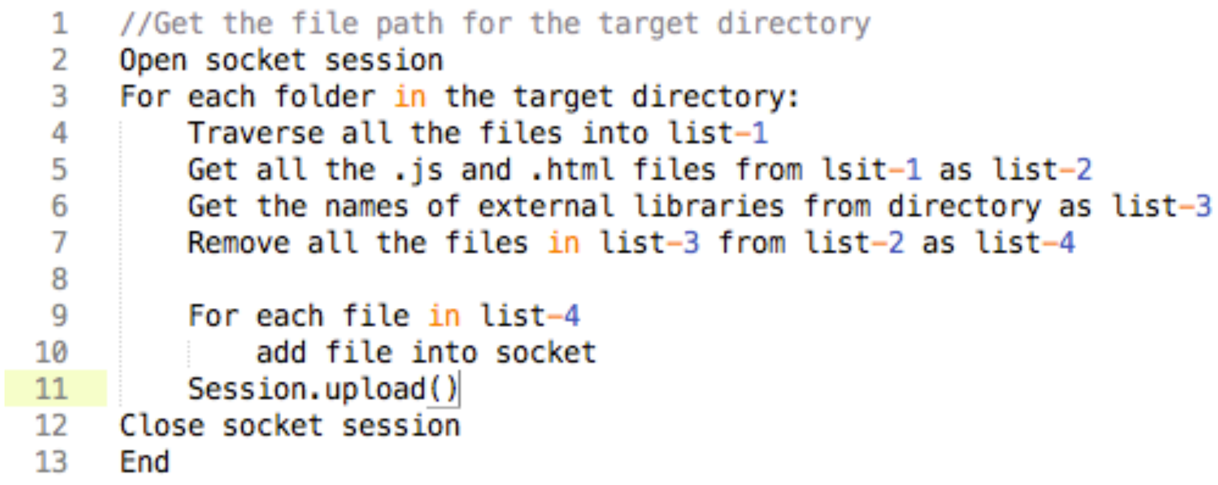
The advantages and disadvantages of different approaches for calculating the code similarity has been developed in the literature review section (Table 2.1).

As the aim of this study is to find practical computational approaches to define similarity, the selected method should be able to go through the source codes in different structures, semantically and textually. It should also be efficient and accurate enough to support the final evaluation of the projects without human support and pre-request. When taking about the mashup development and the context of open data contest into consideration, the solutions in the competition would not be large scale software projects but small scale web-based applications. In order to accomplish these requirements, this study will use MOSS, which is a web-based plagiarism detection tool developed by Stanford University, to measure the similarities among different mashups in the contest. The MOSS system will use local algorisms to calculate the fingerprinting among documents (Schleimer et al., 2003). The basic strategy of the algorism in MOSS system will cover both the semantic and structural approaches to detect similar fragments in the source code (Cosma & Joy, 2012). It could also be learned from the features of different similarity measurement approaches in Table 2.1 that MOSS will be able to meet all the requirements for this study by combining structural and semantic approaches together. Compared to other plagiarism detection tools available online, MOSS is also better at dealing with specific languages such as Java and JavaScript, which is required as the major language for the selected mashups in this study (Hage et al., 2011).

The MOSS system will first develop an abstract syntax tree (AST) generating method to build ASTs from different mashups. While processing the mashup projects collected from Purdue Ironhacks, an AST tree will be built for each mashup project. The MOSS system will recognize the similar ASTs between two projects and apply the semantic similarity measurement in the similar AST blocks (Cosma & Joy, 2012). The measurement of similarity between two projects will be based on the semantic matching in lines in each node structure in the ASTs. And the formula for code similarity in MOSS system is defined as:

$$\text{Similarity} = 2 * S / (2 * S + L + R)$$

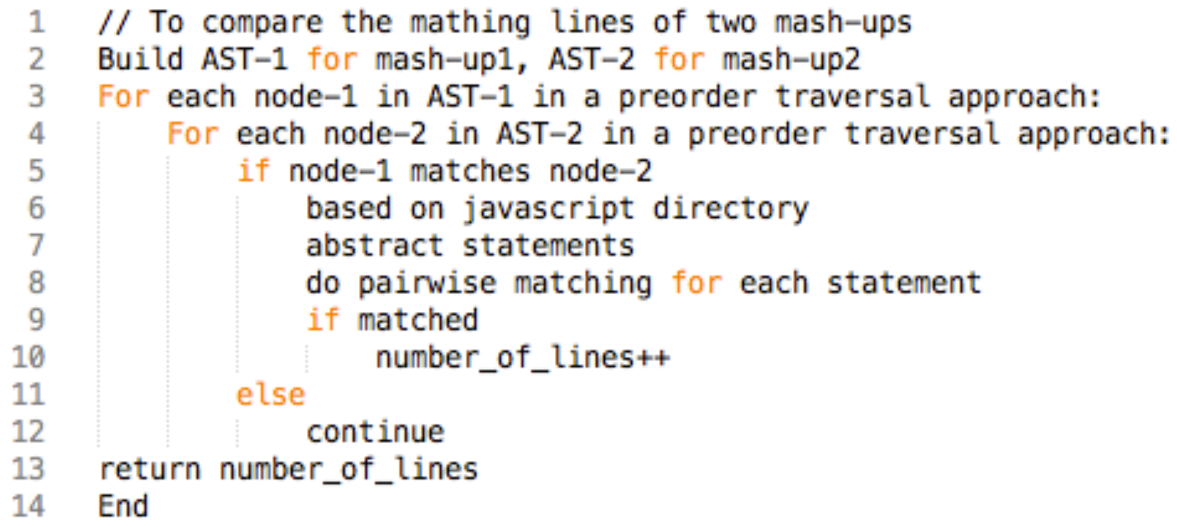
The semantic measurement would be based on the matching algorism of textual, which is the number of lines in MOSS system. This study will develop an interface (Figure 3.1) program to pre-wash the data and send source codes to the MOSS server and get feedback reports from the server. The similarity will be calculated by crawling and analyzing the data from the feedback reports.



```
//Get the file path for the target directory
Open socket session
For each folder in the target directory:
    Traverse all the files into list-1
    Get all the .js and .html files from lsit-1 as list-2
    Get the names of external libraries from directory as list-3
    Remove all the files in list-3 from list-2 as list-4

    For each file in list-4
        add file into socket
    Session.upload()
Close socket session
End
```

Figure 3.1Algorithm to wash data and upload to MOSS



```
// To compare the mathing lines of two mash-ups
Build AST-1 for mash-up1, AST-2 for mash-up2
For each node-1 in AST-1 in a preorder traversal approach:
    For each node-2 in AST-2 in a preorder traversal approach:
        if node-1 matches node-2
            based on javascript directory
            abstract statements
            do pairwise matching for each statement
            if matched
                number_of_lines++
        else
            continue
return number_of_lines
End
```

Figure 3.2 Algorithm for MOSS matching lines detection

3.2.2 Functional feature similarity

Based on the literature review of feature similarity, this study uses the method of set theory and the decision support system theory to measure the functional feature similarity. The functions of the solutions should be divided into separate sets with different characteristics. The weight of matched sub-features will be the similarity result between two projects. This study will go through a scenario-based checking to find out all the related requirement components in mashups and find decision parameters by figuring out the specific tokens in the source code. These components will be used for building structured clusters to support the computational measurement and an overall score for the whole mashup. The parameters will be used to calculate the sub-similarities for different components.

In order to operate the feature separating process automatically instead of subjectively, this study will use the pattern matching based on user interface tokens in the source code for each project. By learning from the theme and requirements of the competition, there will be three major components in each mashup: Google Map interfaces, data visualization graphs and interaction HTML forms (Figure 3.3). For each major component, parameters will be collected to perform the whole functionality for the component. A dictionary for all the potential parameter tokens is built to support the functionality parameter matching in the computational approach. The dictionary is built based on all the user interface tokens for Google Map, three required data visualization JavaScript libraries in the contest and 5 other commonly used front-end development JavaScript libraries (see appendix).

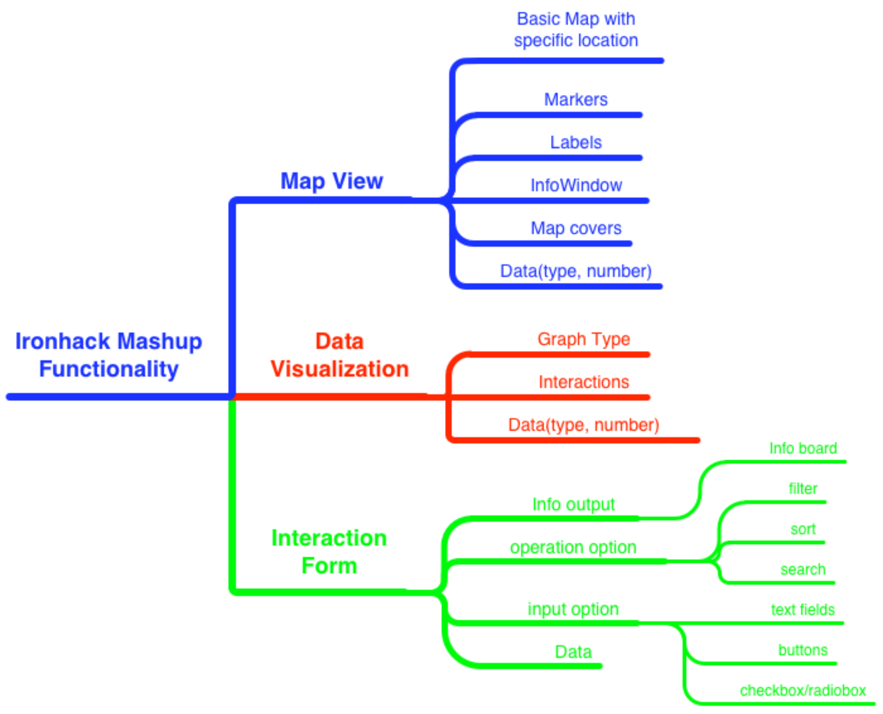


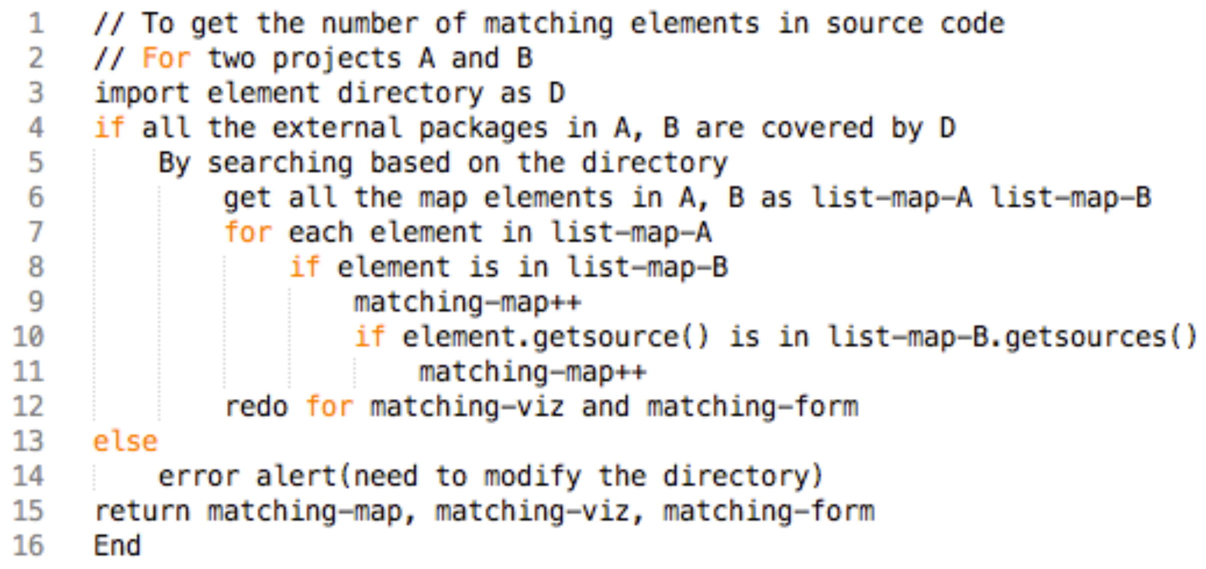
Figure 3.3 Ironhack mashup components

A program will be developed to parse all the JavaScript and HTML contents in those structured mashups to find out the matching of patterns among different projects. The result of feature similarity between two projects will be shown as the percentage of matched patterns in the decision parameters. As the parameters all come from web programming languages, they could be equally weighted in the same kind of web applications or mashups based on the software engineering theory. By learning from the Tversky index measurement (1977) for equally weighted parameters, the formula could be defined as:

$$\text{Similarity} = 2 * S / (2 * S + L + R)$$

S represents the total amount of decision variables in two projects, L represents the number of different variables in the first project and R stands for the number of different variables in the second project.

To ensure the validity of this practical approach, this study will enroll example test cases (see Appendix) on the algorithm to measure the basic functionality parameters in the source code. The 10 test cases cover most of the potential development cases in mashup development in Ironhack including the map development, visualization design and other basic user interactions. Human coding is used for detecting all the matching parameters in these cases and the results would be compared with the parameters detected by the computational algorithm.



```
// To get the number of matching elements in source code
// For two projects A and B
import element directory as D
if all the external packages in A, B are covered by D
    By searching based on the directory
        get all the map elements in A, B as list-map-A list-map-B
        for each element in list-map-A
            if element is in list-map-B
                matching-map++
                if element.getsource() is in list-map-B.getsources()
                    matching-map++
        redo for matching-viz and matching-form
else
    error alert(need to modify the directory)
return matching-map, matching-viz, matching-form
End
```

Figure 3.4 Algorithm for functionality feature matching

3.3 Data Analysis Plan

The processed data is collected from the data collection process by applying different evaluation approaches to evaluate the different dimensions. In the measurement of code similarity, the accuracy and validity of results are guaranteed by the previous researches on the MOSS system. And in the functional feature similarity measurement, we use all the test-case approach to ensure the triangulation and validation of the algorithm.

3.4 Pilot Testing

Before running the research design iteration, there was a pilot test on the research design methodology with pilot data of 22 participants enrolled in the competition to help improving the overall process of the algorithms design. Ten of the participants finally met all the requirements of the contest and became eligible to be considered for the final prizes. This study collected their project source codes in the ends of the four different phases and did a pilot testing on the algorithms developed for the similarity measurement.

3.4.1 Technical Similarity

The MOSS interface developed in this study worked well with the MOSS system server and similarity data were collected in Figure 3.5. To ensure the validity of the technical similarity measurement approach, the data washing process will strictly follow the pre-conditions of MOSS system, which is proven to be a validated public service according to the literature review section. The maximum number of lines in a single file is 270, which means that all of the target files could be regarded as small scale files. According to Figure 3.1, since the external libraries have been removed, the projects should be read to be processed by both the code similarity program and the functional feature similarity program.

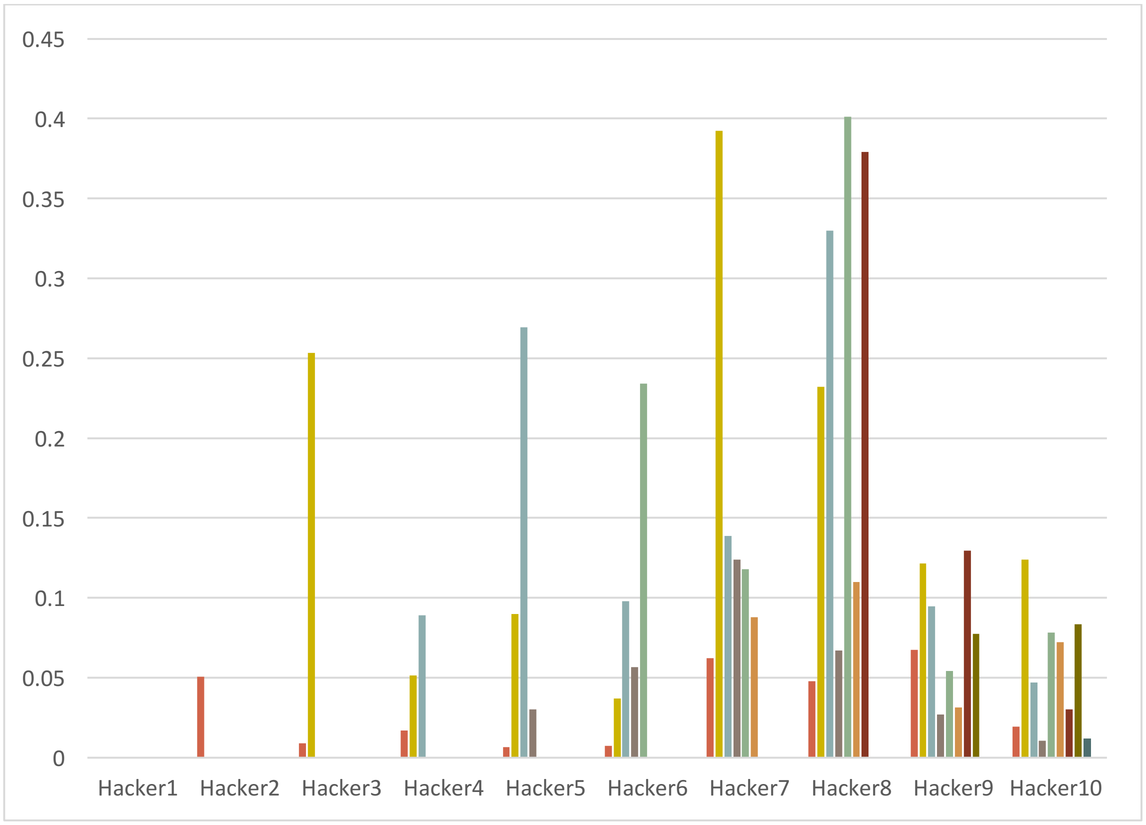


Figure 3.5 Histogram for code similarity – pilot

Figure 3.5 presents the results of similarity calculated based on MOSS system response for the data collected from BlueIronhack. The tested data group size is 10 and pairwise scores are collected. We can learn from the results that the highest similarity score is 0.40 between hacker5 and hacker8. The lowest score is less than 0.01between the mashup projects of hacker1 and hacker5. Among all the technology similarity results, hacker1 and hacker10 has the first two lowest average pairwise similarity scores with the others which means these two participant did not reuse others' code or get reused by others. Hacker8 receives the highest average similarity scores which means his project is most reused by others. The mean of the similarity scores is 0.10815 and the median is

0.077737, where it can be found that the result is a right-skewed distribution. Most of the participants have low code similarity scores with others since they are developing their own souce code to complete their projects. The standard deviation is 0.106067, which means most of the similarity scores are located below 0.4.

3.4.2 Functional Feature Similarity

In the pilot test, this study also performs measurements on the functional feature similarity among mashups in three major components. In Figure 3.6, three major components are equally weighted as they are all top-level components in the mashups in the Ironhack.

To assess the validity of the functional feature similarity measurement algorithm, this study enrolls a test-case approach to test the matching algorithm with well-design test cases. Test cases are developed based on all the scenarios estimated to happen during the life circle of a specific software or information system (Tung & Aldiwan, 2000). If test cases are passed in a significant level, the target software could be regarded as valid in the scenarios of test cases (Zhu et al., 1997). As discussed in Section 3.1, the context for this open data contest is developing map and visualization graph together as mashups to assist end-users. Ten test cases are designed to all the common usage situations and ensure the validity of the algorithm. The result from test cases processing shows that the algorithm is valid for all the scenarios covered by the test cases (Table 3.1).

Table 3.1 Results of test cases processing

Test Case	Desctiption	Designed Parameters	Detected Parameters
1	map	25	25
2	sigma	9	9
3	arbor	13	13
4	d3	200	200
5	layouts	15	15
6	library layouts	200	200
7	viz combination	220	220
8	map combination	33	33
9	layout combination	300	300
10	overall combination	400	400

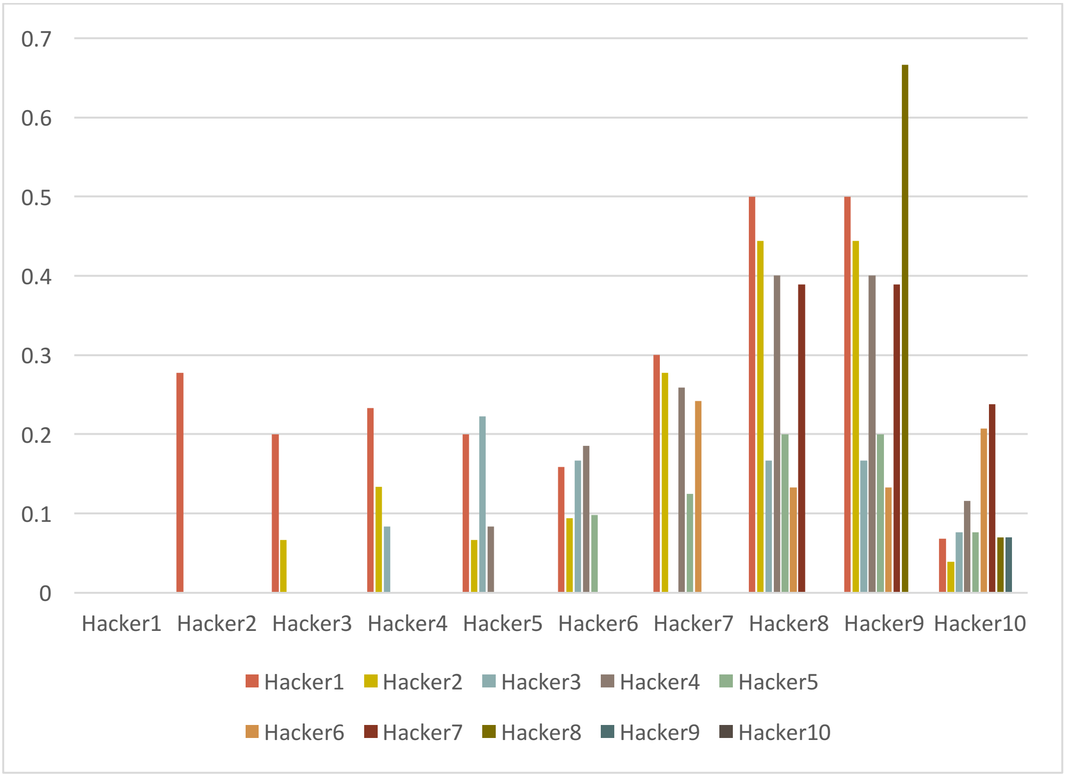


Figure 3.6 Histogram for functional feature similarity - pilot

Figure 3.6 presents the results of overall functional feature similarity scores for the data collected from BlueIronhack. The tested data group size is 10 and pairwise scores are collected. We can learn from the results that the highest similarity score is 0.66

between participants “Hacker9” and “Hacker8”. Among all the functional feature similarity results, participant “Hacker10” has the lowest average pairwise similarity scores with the others which means this participant did not reuse others’ code or get reused by others. Participant “Hacker9” receives the highest average similarity scores which means his project is most reused by others. The mean of the similarity scores is 0.21250 and the median is 0.18519, where it can be found that the result is a right-skewed distribution. Most of the participants have low functional feature similarity scores with others since they are developing their own end-user functions to complete their projects. The standard deviation is 0.14743, which means most of the similarity scores are located below 0.65.

CHAPTER 4. RESULTS AND FINDINGS

In this chapter, this study first reviews the statement of problem to start the data collection and analysis process. All the data for supporting the computational approaches is collected from the open data contest called Green Ironhack based on the specific experiment settings. Before data analysis, data washing is applied and there is a description of the collected data. The similarity measurement approaches for both technology and functionality feature are defined in chapter three and tested in the pilot testing section. The technology similarity is measured using MOSS public service, which is discussed as a fit for the algorithm design methodology in this study. The functional feature similarity is measured based on the decision parameters in each ontology component in the mashups and calculated under the formula of Tversky Index similarity. At the end, this chapter presents the conclusions on the similarity results in the open data innovation contest and describe the further research areas after this study.

4.1 Review of the Problem

Given the fact that the term of "reuse" is becoming more and more important in open data contests with a transparent context, this study is looking forward to an approach to figure out the reuse status by measuring how similar are two mashups in the contest. Rodriguez and Chinea (1998) declared that there are three different dimensions in the evaluation of mashups: technology, functionality feature and visualized feature, which provides the guideline for similarity measurement dimensions. Although there is abundant amount of designs for measuring technology similarity in terms of the source code, no one is proved to be appropriate to the code similarity of data-oriented mashups. One of the goals of this study is to find a specific approach to measure the code similarity in the context of open data contest. The typical features for the most common code similarity measurement approaches are listed in chapter three. In order to fit in the characteristics of the source code in the mashups from the collected project data, this study will choose an open service for code plagiarism detection developed by Stanford University. At the mean time, this study needs also to find out a computational approach to measure the functional feature similarity among different mashups in the open data contest. Human coding, or subjective judging, is the most common method for functionality evaluation and measurement. However, it would benefit more if there is a computational approach to save human cost and provide technical supports. The logic model to measure the functional feature similarity is based on the decision support system. By linking decision parameters in mashups to the tokens in the source code, this study would be able to analyze the functionality in different components based on the parameters found there. Based on the diagram developed by Nikerson and Corter (2008), this study could measure the similarities for different ontology components and finally combine them as an overall similarity score. The aim of this study is to find working computational methods in the specific research settings.

4.2 Research Design Background and Data Overview

The contest that is used to support this study is called "Green Ironhack". There are 26 participants in the contest and all of them are eligible for the final prize evaluation after four rounds of submissions. After checking into the structures and source code of the projects, it shows that all of the 26 projects are using JavaScript as their major language as required by the design methodology. They all use Google Map API as the interface to create map view in their applications and they use one of the three recommended libraries to add visualization graphs into their applications. As for the interaction form developed in HTML language, some of the participants are using external libraries such as bootstrap or angularJS, whose tokens are all included in the dictionaries for functional feature similarity measurement. The external libraries themselves should be removed before analyzing to prevent code overlapping on those libraries.

JavaScript file (.js) and HTML file (.html) are the only two types of file to be used for analysis on similarities. When using file parsing scripts to check into these files, it is found that generally each project will contain two to twenty such files in total. The maximum number of lines in a single file is 354, which means that all of the target files could be regarded as small scale files. According to Table 2.1, since the external libraries have been removed, the projects should be read to be processed by both the code similarity program and the functional feature similarity program.

4.3 Code Similarity Measurement

Figure 4.1 presents the results of similarity calculated based on MOSS system response for the data collected from GreenIronhack. The tested data group size is 26 and pairwise scores are collected. We can learn from the results that the highest similarity score is 0.38 between participant “Hacker11” and participant “Hacker13”. Among all the technology similarity results, participant “Hacker21” and participant “Hacker22” has the first two lowest average pairwise similarity scores with the others which means these two participant did not reuse others’ code or get reused by others. Participant “Hacker21” receives the highest average similarity scores which means his project is most reused by others. The mean of the similarity scores is 0.01488 and the median is 0.001068, where it can be found that the result is a right-skewed distribution. Most of the participants have low code similarity scores with others since they are developing their own souce code to complete their projects.The standard deviation is 0.036591, which means most of the similarity scores are located below 0.12.

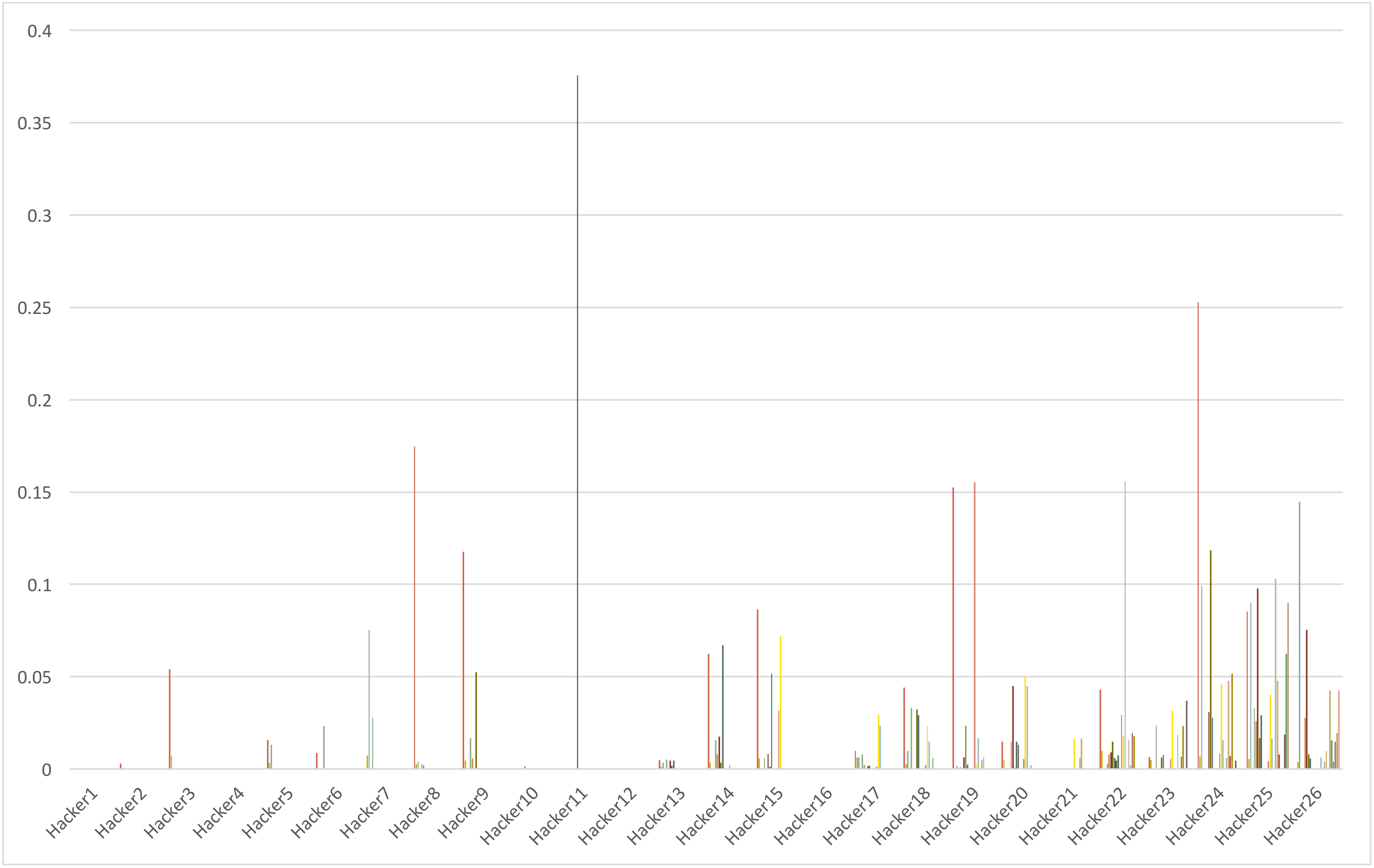
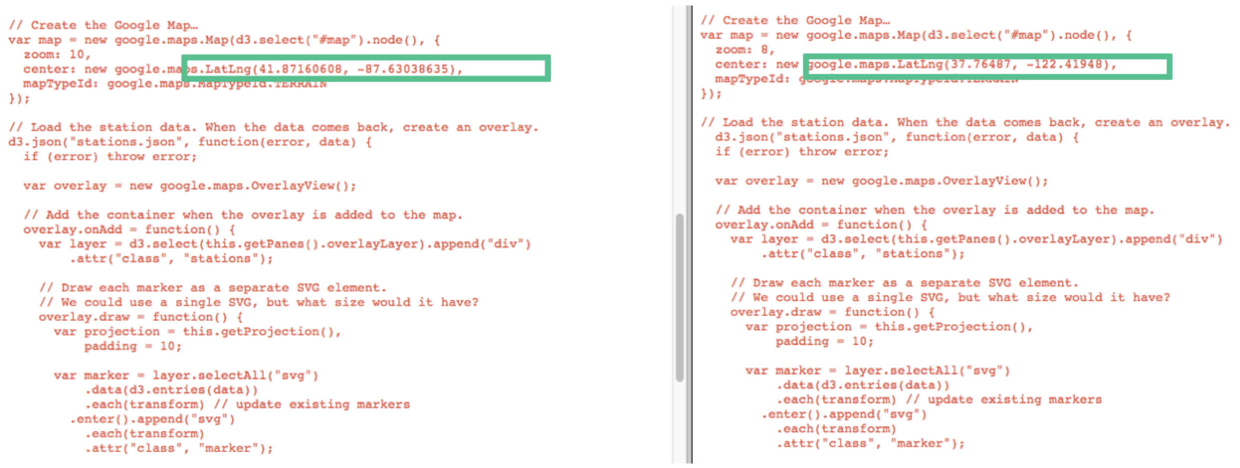


Figure 4.1 Histogram for code similarity

When look into the two participants with the highest pairwise similarity score, we can get the feedback of code similarity from MOSS system (Figure 4.2). The code similarity measurement report shows that the two participants are using the same source code fragments to develop Google Map components, which leads to high similarity in technology dimension. The only part where they are different is the basic geolocation data of map.



```
// Create the Google Map…
var map = new google.maps.Map(d3.select("#map").node(), {
  zoom: 10,
  center: new google.maps.LatLng(41.87160608, -87.63038635),
  mapTypeId: google.maps.MapTypeId.TERRAIN
});

// Load the station data. When the data comes back, create an overlay.
d3.json("stations.json", function(error, data) {
  if (error) throw error;

  var overlay = new google.maps.OverlayView();

  // Add the container when the overlay is added to the map.
  overlay.onAdd = function() {
    var layer = d3.select(this.getPanes().overlayLayer).append("div")
        .attr("class", "stations");

    // Draw each marker as a separate SVG element.
    // We could use a single SVG, but what size would it have?
    overlay.draw = function() {
      var projection = this.getProjection(),
          padding = 10;

      var marker = layer.selectAll("svg")
          .data(d3.entries(data))
          .each(transform) // update existing markers
        .enter().append("svg")
          .each(transform)
          .attr("class", "marker");
```

```
// Create the Google Map…
var map = new google.maps.Map(d3.select("#map").node(), {
  zoom: 8,
  center: new google.maps.LatLng(37.76487, -122.41948),
  mapTypeId: [illegible]
});

// Load the station data. When the data comes back, create an overlay.
  d3.json("stations.json", function(error, data) {
  if (error) throw error;

  var overlay = new google.maps.OverlayView();

  // Add the container when the overlay is added to the map.
  overlay.onAdd = function() {
    var layer = d3.select(this.getPanes().overlayLayer).append("div")
        .attr("class", "stations");

    // Draw each marker as a separate SVG element.
    // We could use a single SVG, but what size would it have?
    overlay.draw = function() {
      var projection = this.getProjection(),
          padding = 10;

      var marker = layer.selectAll("svg")
          .data(d3.entries(data))
          .each(transform) // update existing markers
        .enter().append("svg")
          .each(transform)
          .attr("class", "marker");
```

Figure 4.2 Code similarity result from MOSS system

4.4 Functional feature Similarity Measurement

Figure 4.3 presents the results of overall functional feature similarity scores for the data collected from GreenIronhack. The tested data group size is 26 and pairwise scores are collected. We can learn from the results that the highest similarity score is 0.3 between participant “Hacker12” and “Hacker13”. Among all the results for functional feature similarity, participant “Hacker22” has the lowest average pairwise similarity scores with the others which means this participant did not reuse others’ code or get reused by others. Participant “Hacker21” receives the highest average similarity scores which means his project is most reused by others. The mean of the similarity scores is

0.123984 and the median is 0.117484, where it can be found that the result is a right-skewed distribution. Most of the participants have low functional feature similarity scores with others since they are developing their own end-user functions to complete their projects.The standard deviation is 0.046787, which means most of the similarity scores are located below 0.26

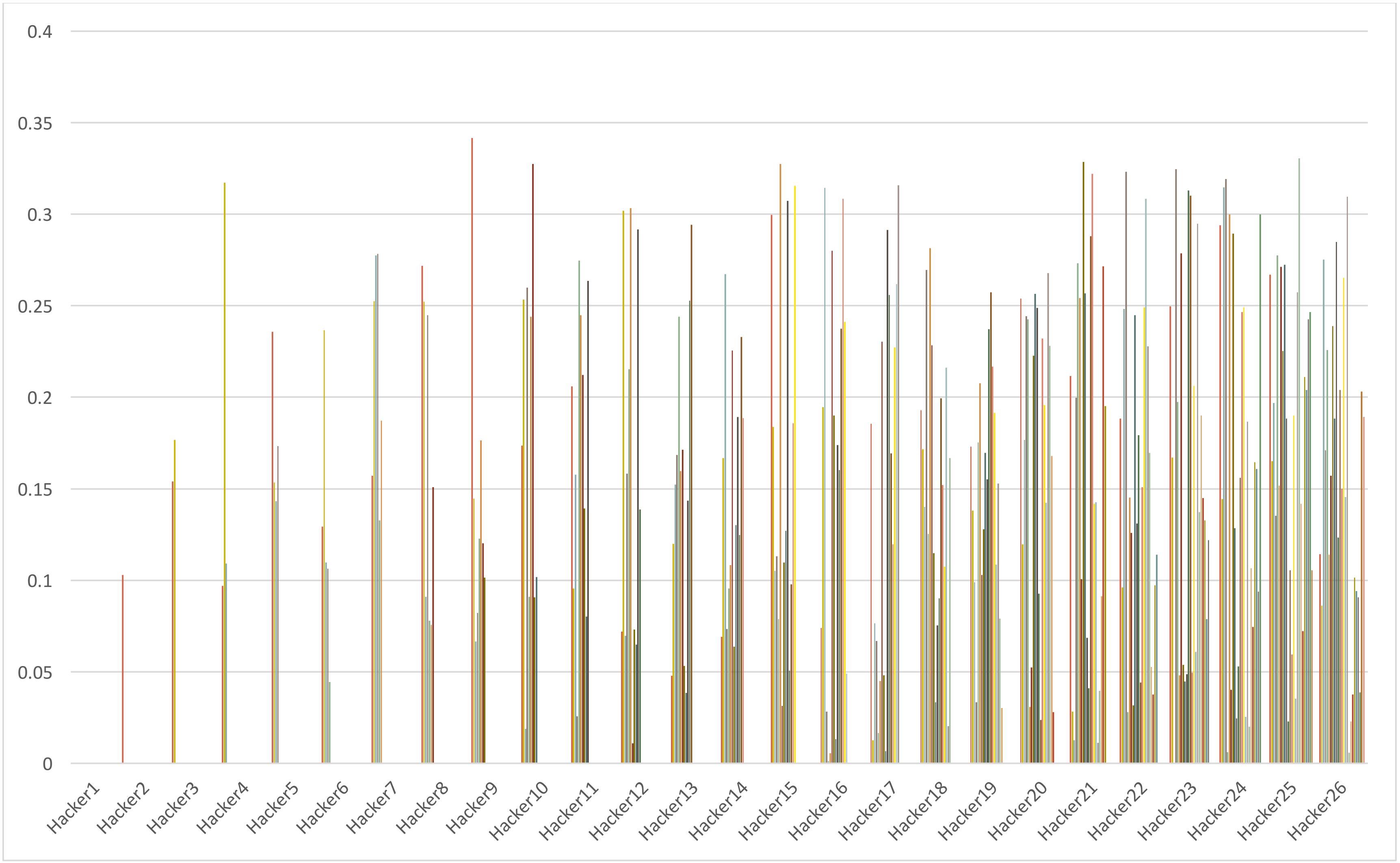


Figure 4.3 Histogram for map components similarity

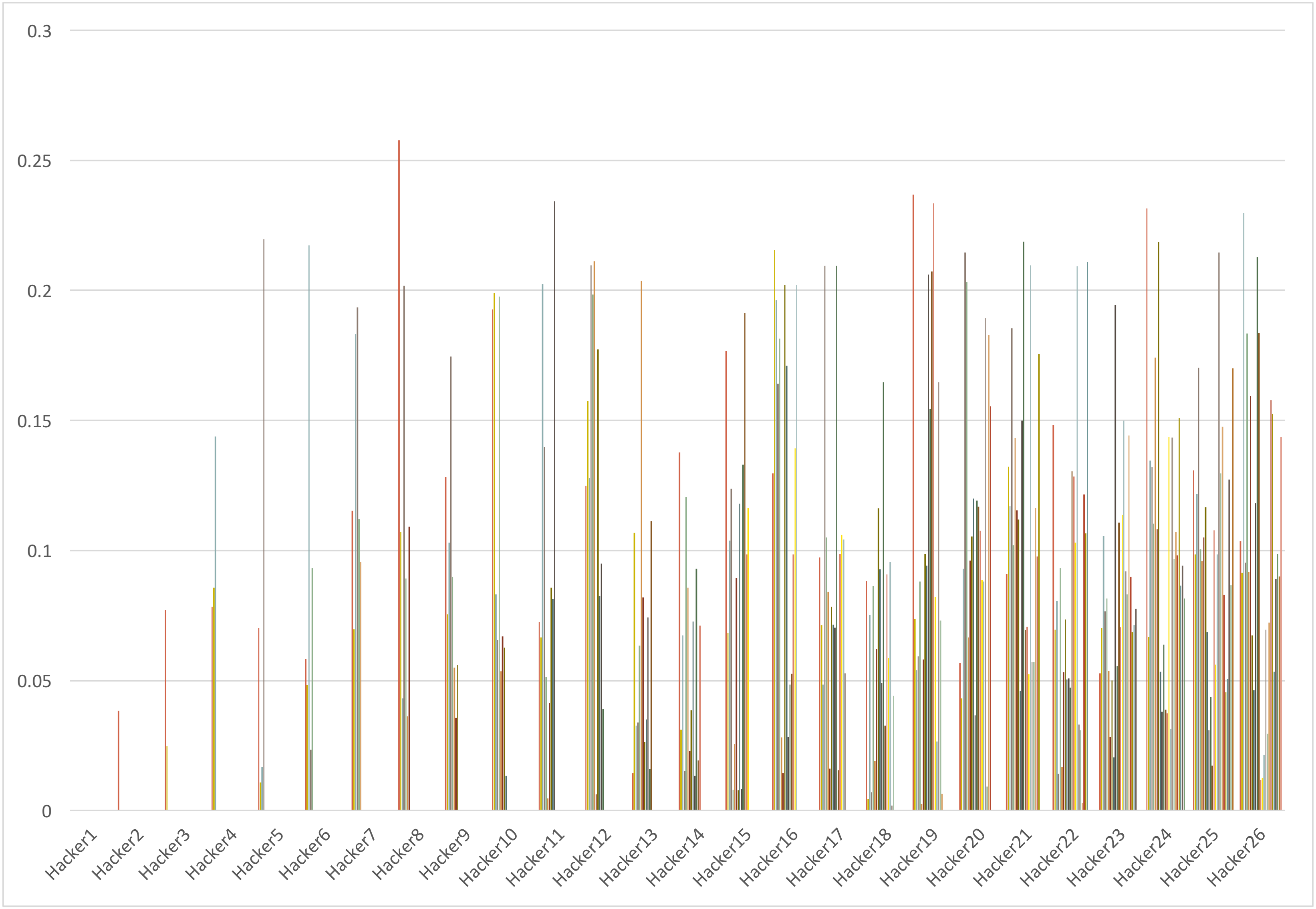


Figure 4.4 Histogram for visualized graph components similarity

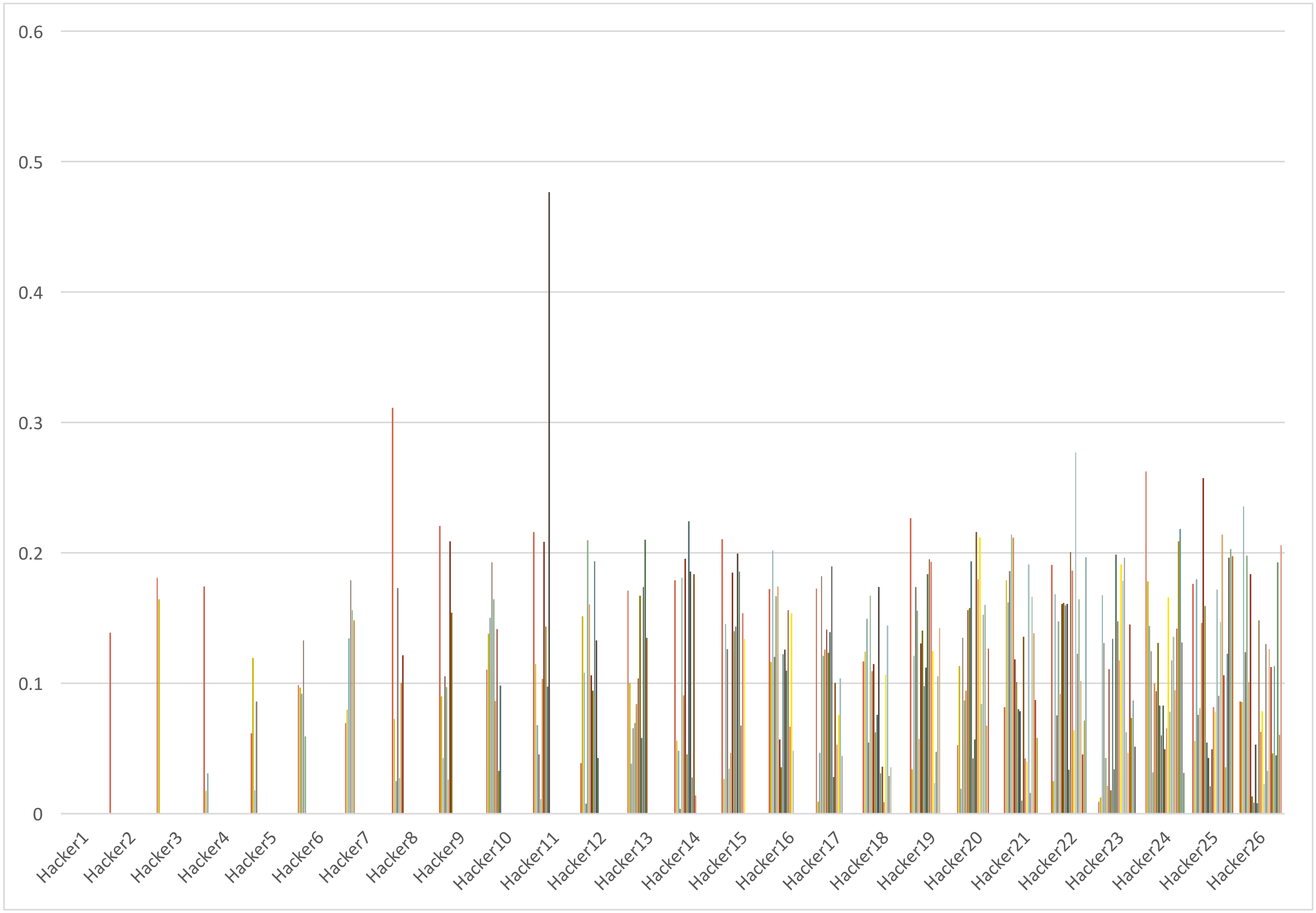


Figure 4.5 Histogram for interaction form components similarity

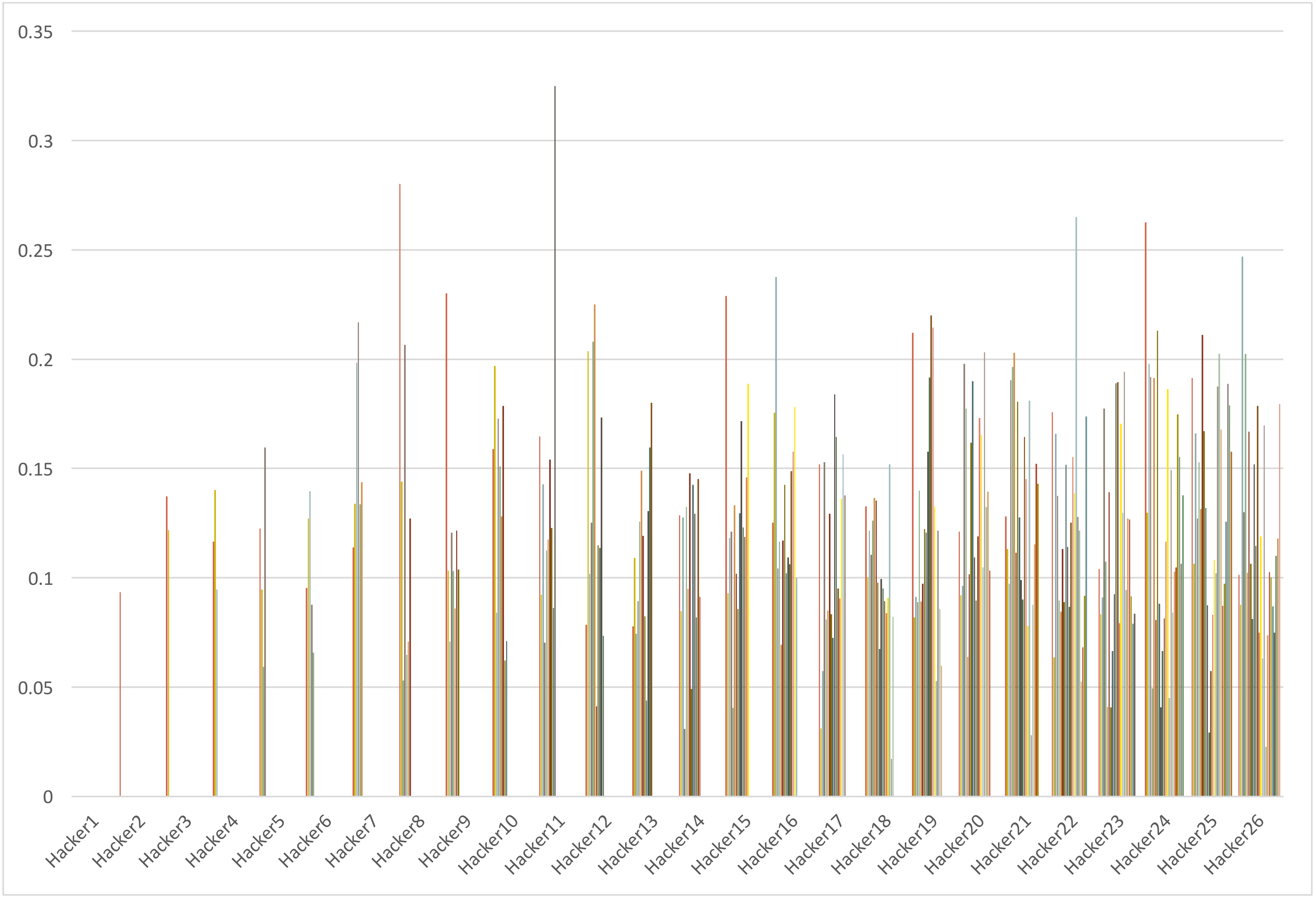


Figure 4.6 Histogram for overall functional feature similarity

When look into the two participants with the highest functional feature similarity score, we can find there are several similar sub-components for end-user functions (Figure 4.4). It shows that for the google map components, the two participants are both developing a basic map for a certain city: one is San Francisco and the other is Chicago. They are also using the same markers and layout strategies to show the location information on the maps, which leads to high similarity in functional feature dimension.

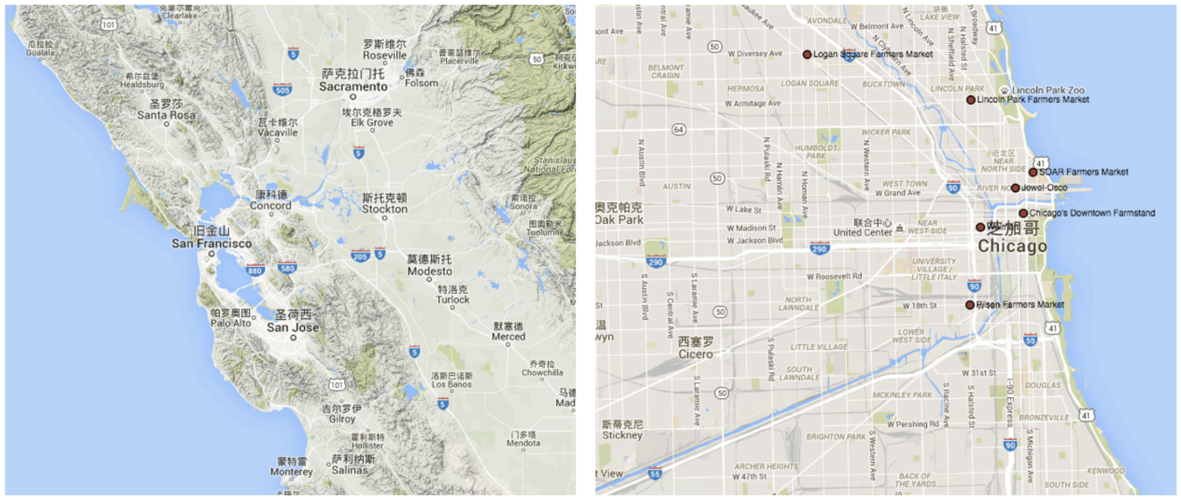


Figure 4.7 Projects with highest functional feature similarity score

4.5 Discussion

In this study, the framework has been developed that there are three-level dimensions in the field of mashup evaluation and similarity measurement (Rodriguez & Chinea, 1998). The problem has been raised in the beginning of the study that measuring the reuse activity is significant for both practical and theoretical progress in the context of transparent open data contest (Nickerson, 2014). We have also developed novel approaches for mashup similarity measurement in both source code level and the functionality feature level.

The proposed method for technical similarity between the source codes of two mashups is based on the structure and semantic matching approaches in the field of computer software. That is, the algorithm uses both abstract syntax tree (AST) and semantic meaning of statements. The AST that provides local-level structural view is included in the parse tree (Baxter et al., 1998). In order to compare the parse trees and process the semantic meaning, this study enrolled the open public service called MOSS to detect the matching results. MOSS is validated as the tool to complete the similarity matching for the context of open data contest (Schleimer et al., 2003). Finally, the proposed method will collect data from MOSS results and calculate the pairwise similarity scores (Baxter et al., 1998).

The method for functional feature similarity measurement is based on the parameter matching in two target projects (Özacar, 2016). Supported by decision system theory, a dictionary has been built to cover all the potential parameters and tokens for a target software and the algorithm will calculate the numbers of matching items in two mashup projects. After getting all the matching numbers from the projects, the algorithm will apply a Tversky similarity measurement formula to calculate the final pairwise similarity (Tversky, 1977).

In the pilot test case of BlueIronhack and the practical case of GreenIronhack, it is shown that the proposed measurement methods could work properly with the given source codes from the mashup projects. In particular, the algorithms also passed all the designed test cases and worked well with all the pairwise scores for participants in a contest (Tung & Aldiwan, 2000).

One advantage of the algorithms designed in this study is that they are designed for the specific context of open data contest and could achieve higher accuracy (Yamamoto et al., 2005). The algorithms are all focusing on JavaScript projects with small scale of source code files and can get rid of the effect of external libraries. Since the MOSS system can support multiple different programming languages, the code similarity measurement method could easily be transformed to detect other kinds of programming languages other than JavaScript. Once the parameter dictionary is changed for a specific language, the functionality feature measurement method is also suitable to multiple programming languages.

4.6 Contributions and Limitations

This study makes three major contributions. First, it presents a theoretically grounded conceptualization of similarity for mashups in open data contest that extends the existing literature on code similarity in software engineering (Yu et al., 2008). Instead of the original similarity definitions for large-scale or enterprise softwares, this study raises three different dimensions, technical similarity, functional feature similarity and visualized feature similarity for the similarity measurement for mashups developed in open data contest (Yamamoto et al., 2005).

Second, it theoretically develops a computational code similarity measurement approach for mashups in the context of open data contest. Based on the literature review, the study chooses MOSS system to perform both structural and semantic matching for similarity measurement, which have been developed already for general software similarity measurement, to fit all the key characteristics of the open data innovation contest (Hage et al., 2011). The third contribution of this study is to define the theoretical conceptualization of functional feature similarity as well as develop the computational method for it. It advances the literature on cognitive psychology by providing computational approach to the conceptual methodology. Instead of focusing on the source-code functions in the field of computer science, this study is more focusing on the end-user functions with a user perspective by linking code with end-user features. The concept of functional feature similarity is developed based on the ontology components theory of Beydoun (2014). And the components are generated by finding the scenario-based parameters and cluster of parameters (Nikerson & Corter, 2008). This study also involves the theory of decision support system to introduce the concept of decision parameters, which is supported by the literature and can provide the weighting strategy for parameters in calculation. The DSS theory is used to support the similarity measurement algorithm (Özacar, 2016).

As for the limitations, this study is currently focusing on two of the three dimensions of mashup similarity measurement, which are code similarity and the functional feature similarity. The generalizability in this study is limited and only applies to the study with 100 mashup projects developed during Purdue GreenIronhack. The validation of the computational approaches for functional feature similarity is only tested on the designed test cases for the mashups in open data innovation contest. The design settings for this study is restricted to the specific web programming language of JavaScript with limited external libraries for the mashups developed in the open data contest.

4.7 Future Research

Upon the conclusion of this study, several areas of future research can be addressed. The future research should cover the measurement of visualized feature similarity, which is another important dimension of similarity for mashups in open data contest (Yamamoto et al., 2005). The information visualization feature can be beneficial to the process of learning and decision making (Zhang & Whinston, 1995). The correlations of similarities between different dimensions can also be worth further analysis and discussion. The result of correlations would be helpful to learn the reuse activities and transparency in different dimensions.

In order to measure the amount of reuse, we should have further development on the similarity measurement algorithms to get the actual reuse of participants in a transparent open data contest. In the context of Purdue Ironhack project, we assume that project A is the first stage solution for participant X, project B is the first stage solution for participant Y and project C is the second stage solution for participant X (Figure 4.1). The current similarity measurement algorithms in this study is measuring the overlapping field: $A \cap B = 2AB/(A+B+2AB) = AB + ABC$. The actual reuse of X reuse Y's code is the field of: $(C-A) \cap B = BC$. Instead of doing a pairwise similarity measurement, further study needs to get the normalized similarity result to find the intersection part of ABC to get the actual reuse amount in a reuse activity.

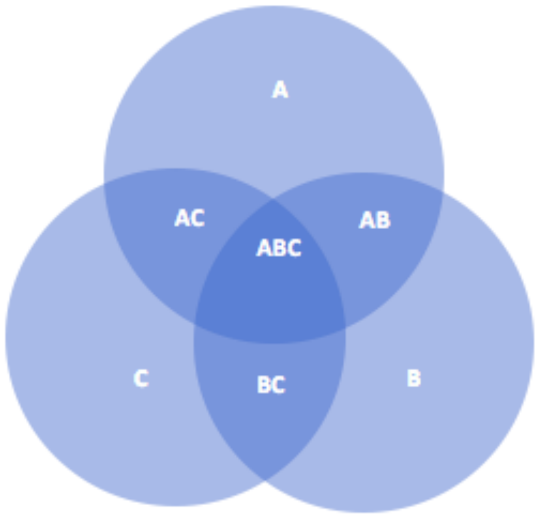


Figure 4.8 Mashup reuse in Purdue Ironhack

4.8 Summary and Concluding Remarks

In this chapter, I apply the designed computational methods for similarity measurement to the data collected from Purdue GreenIronhack to calculate the code similarity and functional feature similarity of the mashups in that contest. This chapter also presents the discussions to the primary research questions posted in the first section. It summaries the conceptualization on the similarity of mashups developed in open data contest. It also reviews the design and validation test of the computational approaches for similarity measurement. In conclusion, this research sets the stage for future research on open data contests. There is much tremendous opportunity to create novel applications from open data that re-use existing components, and I hope that others will build upon this research to advance theory and practice.

LIST OF REFERENCES

LIST OF REFERENCES

Antoniol, G., Villano, U., Merlo, E., & Di Penta, M. (2002). Analyzing cloning evolution in the Linux kernel. Information and Software Technology, 44(13), 755–765. http://doi.org/10.1016/S0950-5849(02)00123-4

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). DBpedia: A Nucleus for a Web of Open Data. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, … P. Cudré-Mauroux (Eds.), The Semantic Web (pp. 722–735). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-76298-0_52

Baker, B. S. (1995). On finding duplication and near-duplication in large software systems. In , Proceedings of 2nd Working Conference on Reverse Engineering, 1995 (pp. 86–95). http://doi.org/10.1109/WCRE.1995.514697

Basit, H. A., & Jarzabek, S. (2009). A Data Mining Approach for Detecting Higher-Level Clones in Software. IEEE Transactions on Software Engineering, 35(4), 497–514. http://doi.org/10.1109/TSE.2009.16

Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M., & Bier, L. (1998). Clone detection using abstract syntax trees. In , International Conference on Software Maintenance, 1998. Proceedings (pp. 368–377). http://doi.org/10.1109/ICSM.1998.738528

Berndt, D. J., & Watkins, A. (2004). Investigating the performance of genetic algorithm-based software test case generation. In Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings (pp. 261–262). http://doi.org/10.1109/HASE.2004.1281750

Bildhauer, D., Horn, T., & Ebert, J. (2009). Similarity-driven Software Reuse. In Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models (pp. 31–36). Washington, DC, USA: IEEE Computer Society. http://doi.org/10.1109/CVSM.2009.5071719

Boudreau, K. J., Lacetera, N., & Lakhani, K. R. (2011). Incentives and Problem Uncertainty in Innovation Contests: An Empirical Analysis. Management Science, 57(5), 843–863. http://doi.org/10.1287/mnsc.1110.1322

Boudreau, K. J., & Lakhani, K. R. (2015). "Open" disclosure of innovations, incentives and follow-on reuse: Theory on processes of cumulative innovation and a field experiment in computational biology. Research Policy, 44(1), 4–19. http://doi.org/10.1016/j.respol.2014.08.001

Brown, R. B. K., Beydoun, G., Low, G., Tibben, W., Zamani, R., García-Sánchez, F., & Martinez-Bejar, R. (2014). Computationally efficient ontology selection in software requirement planning. Information Systems Frontiers, 18(2), 349–358. http://doi.org/10.1007/s10796-014-9540-3

Bullinger, A. C., Neyer, A.-K., Rass, M., & Moeslein, K. M. (2010). Community-Based Innovation Contests: Where Competition Meets Cooperation. Creativity and Innovation Management, 19(3), 290–303. http://doi.org/10.1111/j.1467-8691.2010.00565.x

Cheng, C. C. J., & Huizingh, E. K. R. E. (2014). When Is Open Innovation Beneficial? The Role of Strategic Orientation. Journal of Product Innovation Management, 31(6), 1235–1253. http://doi.org/10.1111/jpim.12148

Chesbrough, H. W. (2006). Open Innovation: The New Imperative for Creating and Profiting from Technology. Harvard Business Press.

Chilowicz, M., Duris, E., & Roussel, G. (2009). Syntax tree fingerprinting for source code similarity detection. In IEEE 17th International Conference on Program Comprehension, 2009. ICPC '09 (pp. 243–247). http://doi.org/10.1109/ICPC.2009.5090050

Comaniciu, D., & Meer, P. (2002). Mean shift: a robust approach toward feature space analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(5), 603–619. http://doi.org/10.1109/34.1000236

Cosma, G., & Joy, M. (2012). An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis. IEEE Transactions on Computers, 61(3), 379–394. http://doi.org/10.1109/TC.2011.223

Ding, L., DiFranzo, D., Graves, A., Michaelis, J. R., Li, X., McGuinness, D. L., & Hendler, J. A. (2010). TWC Data-gov Corpus: Incrementally Generating Linked Government Data from Data.Gov. In Proceedings of the 19th International Conference on World Wide Web (pp. 1383–1386). New York, NY, USA: ACM. http://doi.org/10.1145/1772690.1772937

Ducasse, S., Rieger, M., & Demeyer, S. (1999). A language independent approach for detecting duplicated code. In IEEE International Conference on Software Maintenance, 1999. (ICSM '99) Proceedings (pp. 109–118). http://doi.org/10.1109/ICSM.1999.792593

Edberg, D., Ivanova, P., & Kuechler, W. (2012). Methodology Mashups: An Exploration of Processes Used to Maintain Software. Journal of Management Information Systems, 28(4), 271–304. http://doi.org/10.2753/MIS0742-1222280410

Frank T. Piller, D. W. (2006). Toolkits for Idea Competitions: A Novel Method to Integrate Users in New Product Development. R&amp;D Management, 36(3), 307 – 318. http://doi.org/10.1111/j.1467-9310.2006.00432.x

Galambos, C., Kittler, J., & Matas, J. (2001). Gradient based progressive probabilistic Hough transform. IEE Proceedings - Vision, Image and Signal Processing, 148(3), 158–165. http://doi.org/10.1049/ip-vis:20010354

Hage, J., Rademaker, P., & van Vugt, N. (2011). Plagiarism Detection for Java: A Tool Comparison. In Computer Science Education Research Conference (pp. 33–46). Open Univ., Heerlen, The Netherlands, The Netherlands: Open Universiteit, Heerlen. Retrieved from http://dl.acm.org/citation.cfm?id=2043594.2043597

Henriques, A. (2007). Corporate Truth: The Limits to Transparency. Earthscan.

Lague, B., Proulx, D., Mayrand, J., Merlo, E. M., & Hudepohl, J. (1997). Assessing the Benefits of Incorporating Function Clone Detection in a Development Process. In Proceedings of the International Conference on Software Maintenance (p. 314–). Washington, DC, USA: IEEE Computer Society. Retrieved from http://dl.acm.org/citation.cfm?id=645545.853273

Law, E. L.-C., Roto, V., Hassenzahl, M., Vermeeren, A. P. O. S., & Kort, J. (2009). Understanding, Scoping and Defining User Experience: A Survey Approach. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 719–728). New York, NY, USA: ACM. http://doi.org/10.1145/1518701.1518813

Leite, P. J. C. S. do P., & Cappelli, C. (2010). Software Transparency. Business & Information Systems Engineering, 2(3), 127–139. http://doi.org/10.1007/s12599-010-0102-z

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. Decision Support Systems, 15(4), 251–266. http://doi.org/10.1016/0167-9236(94)00041-2

Mende, T., Koschke, R., & Beckwermert, F. (2009). An Evaluation of Code Similarity Identification for the Grow-and-prune Model. J. Softw. Maint. Evol., 21(2), 143–169. http://doi.org/10.1002/smr.v21:2

Nickerson, J., Corter, J., Tversky, B., Zahner, D., & Rho, Y. (2008). The Spatial Nature of Thought: Understanding Systems Design Through Diagrams. ICIS 2008 Proceedings. Retrieved from http://aisel.aisnet.org/icis2008/216

Nickerson, J. V. (2014). Collective Design: Remixing and Visibility (SSRN Scholarly Paper No. ID 2424866). Rochester, NY: Social Science Research Network. Retrieved from http://papers.ssrn.com/abstract=2424866

Nickerson, J. V., Corter, J. E., Tversky, B., Zahner, D., & Rho, Y. J. (2008). Diagrams as Tools in the Design of Information Systems. In J. S. Gero & A. K. Goel (Eds.), Design Computing and Cognition ’08 (pp. 103–122). Springer Netherlands. Retrieved from http://link.springer.com/chapter/10.1007/978-1-4020-8728-8_6

Nunamaker, J. F., Dennis, A. R., Valacich, J. S., Vogel, D., & George, J. F. (1991). Electronic Meeting Systems. Commun. ACM, 34(7), 40–61. http://doi.org/10.1145/105783.105793

Özacar, T. (2016). A tool for producing structured interoperable data from product features on the web. Information Systems, 56, 36–54. http://doi.org/10.1016/j.is.2015.09.002

Perkmann, M., & Schildt, H. (2015). Open data partnerships between firms and universities: The role of boundary organizations. Research Policy, 44(5), 1133–1143. http://doi.org/10.1016/j.respol.2014.12.006

Ping Zhang, G. M. Y. D. (2000). Satisfiers and Dissatisfiers: A Two-Factor Model for Website Design and Evaluation. Journal of American Society for Information Science, 51(14), 1253–1268. http://doi.org/10.1002/1097-4571(2000)9999:99993.0.CO;2-O

Rodriguez, R., Chinea, G., Lopez, N., Pons, T., & Vriend, G. (1998). Homology modeling, model and software evaluation: three related resources. Bioinformatics (Oxford, England), 14(6), 523–528.

Schleimer, S., Wilkerson, D. S., & Aiken, A. (2003). Winnowing: Local Algorithms for Document Fingerprinting. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (pp. 76–85). New York, NY, USA: ACM. http://doi.org/10.1145/872757.872770

Shadbolt, N., O'Hara, K., Berners-Lee, T., Gibbins, N., Glaser, H., Hall, W., & schraefel, m c. (2012). Linked Open Government Data: Lessons from Data.gov.uk. IEEE Intelligent Systems, 27(3), 16–24. http://doi.org/10.1109/MIS.2012.23

Terwiesch, C., & Xu, Y. (2008). Innovation Contests, Open Innovation, and Multiagent Problem Solving. Management Science, 54(9), 1529–1543. http://doi.org/10.1287/mnsc.1080.0884

Tung, Y.-W., & Aldiwan, W. S. (2000). Automating test case generation for the new generation mission software system. In 2000 IEEE Aerospace Conference Proceedings (Vol. 1, pp. 431–437 vol.1). http://doi.org/10.1109/AERO.2000.879426

Tversky, A. (1977). Features of similarity. Psychological Review, 84(4), 327–352. http://doi.org/10.1037/0033-295X.84.4.327

Weston, J., Ratle, F., Mobahi, H., & Collobert, R. (2012). Deep Learning via Semi-supervised Embedding. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade (pp. 639–655). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-35289-8_34

Yamamoto, T., Matsushita, M., Kamiya, T., & Inoue, K. (2005). Measuring Similarity of Large Software Systems Based on Source Code Correspondence. In F. Bomarius & S. Komi-Sirviö (Eds.), Product Focused Software Process Improvement (pp. 530–544). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/11497455_41

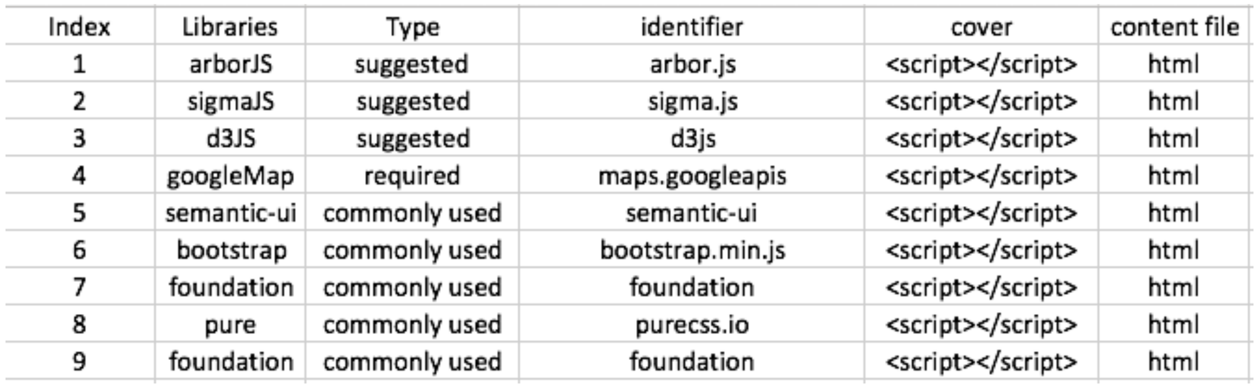
Yu, J., Benatallah, B., Casati, F., & Daniel, F. (2008). Understanding Mashup Development. IEEE Internet Computing, 12(5), 44–52. http://doi.org/10.1109/MIC.2008.114

Zhang, P., & Whinston, A. (1995). Business Information Visualization for Decision-Making Support - A Research Strategy. iSchool Faculty Scholarship. Retrieved from http://surface.syr.edu/istpub/15

Zhu, H., Hall, P. A. V., & May, J. H. R. (1997). Software Unit Test Coverage and Adequacy. ACM Comput. Surv., 29(4), 366–427. http://doi.org/10.1145/267580.267590

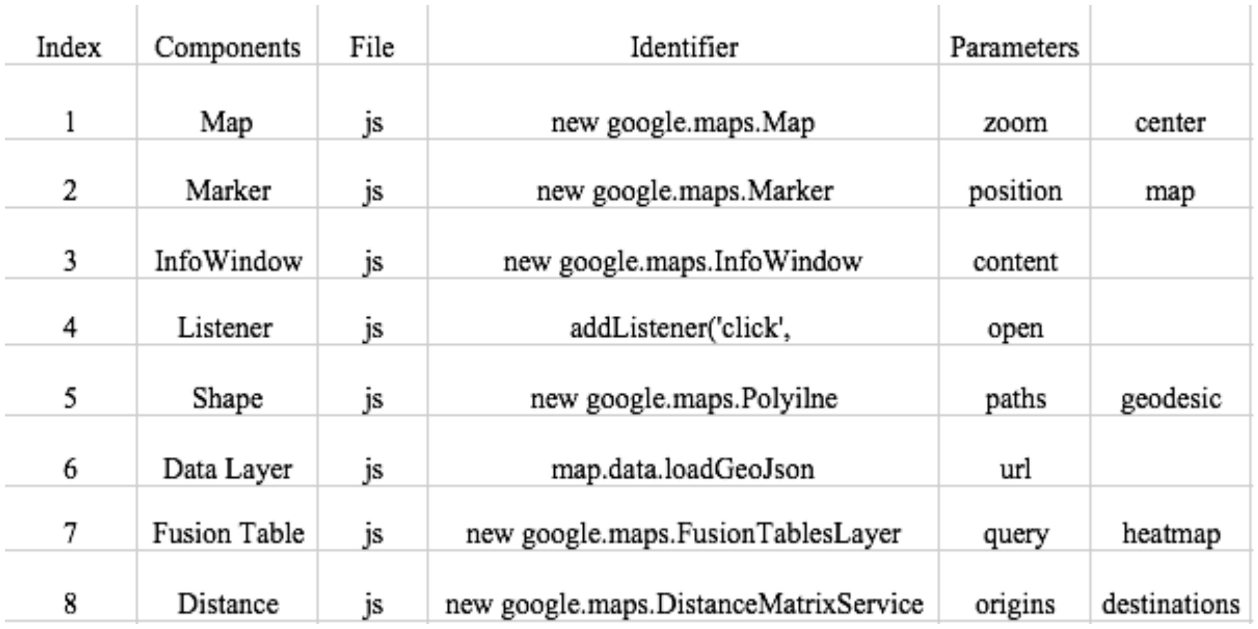
APPENDICES

Appendix A List of Libraries



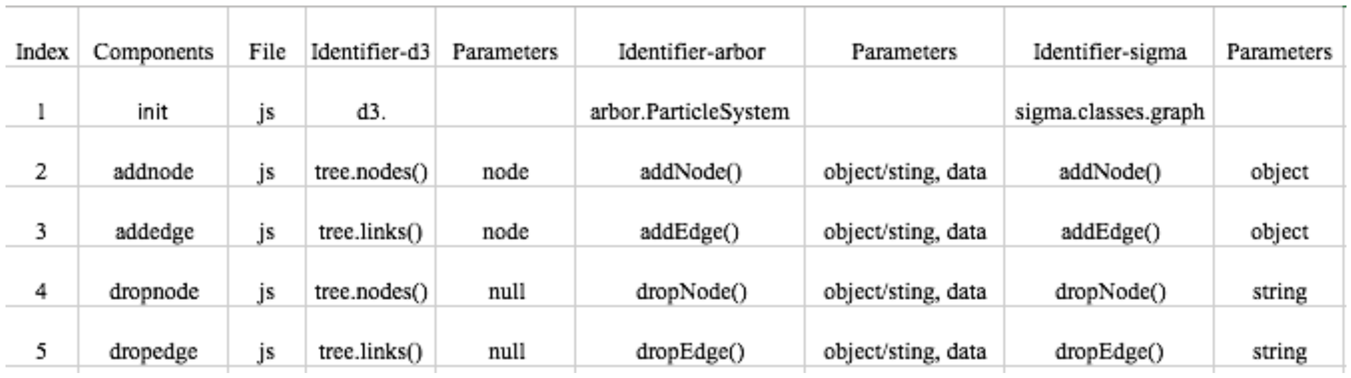
Index	Libraries	Type	identifier	cover	content file
1	arborJS	suggested	arbor.js	<script></script>	html
2	sigmaJS	suggested	sigma.js	<script></script>	html
3	d3JS	suggested	d3js	<script></script>	html
4	googleMap	required	maps.googleapis	<script></script>	html
5	semantic-ui	commonly used	semantic-ui	<script></script>	html
6	bootstrap	commonly used	bootstrap.min.js	<script></script>	html
7	foundation	commonly used	foundation	<script></script>	html
8	pure	commonly used	purecss.io	<script></script>	html
9	foundation	commonly used	foundation	<script></script>	html

Appendix B Dictionary for Google Map



Index	Components	File	Identifier	Parameters	
1	Map	js	new google.maps.Map	zoom	center
2	Marker	js	new google.maps.Marker	position	map
3	InfoWindow	js	new google.maps.InfoWindow	content	
4	Listener	js	addListener('click',	open	
5	Shape	js	new google.maps.Polyilne	paths	geodesic
6	Data Layer	js	map.data.loadGeoJson	url	
7	Fusion Table	js	new google.maps.FusionTablesLayer	query	heatmap
8	Distance	js	new google.maps.DistanceMatrixService	origins	destinations

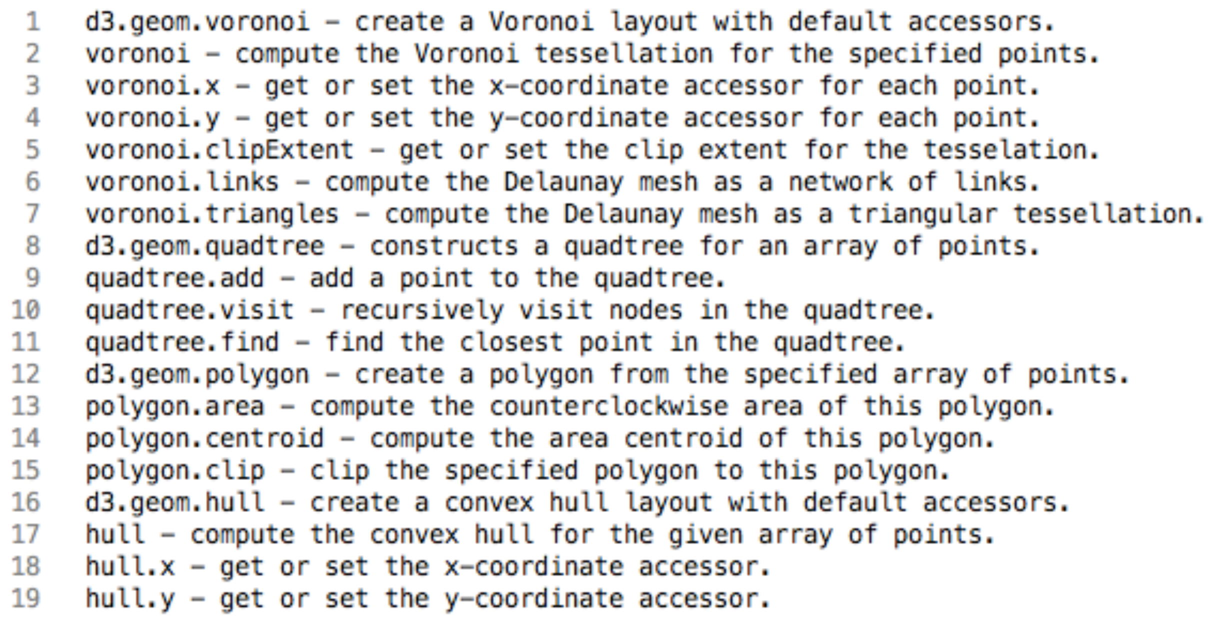
Appendix C Dictionary for Graph-style Visualization Libraries



Index	Components	File	Identifier-d3	Parameters	Identifier-arbor	Parameters	Identifier-sigma	Parameters
1	init	js	d3.		arbor.ParticleSystem		sigma.classes.graph	
2	addnode	js	tree.nodes()	node	addNode()	object/sting, data	addNode()	object
3	addedge	js	tree.links()	node	addEdge()	object/sting, data	addEdge()	object
4	dropnode	js	tree.nodes()	null	dropNode()	object/sting, data	dropNode()	string
5	dropedge	js	tree.links()	null	dropEdge()	object/sting, data	dropEdge()	string

Appendix D Examples from the Dictionary for D3 Library

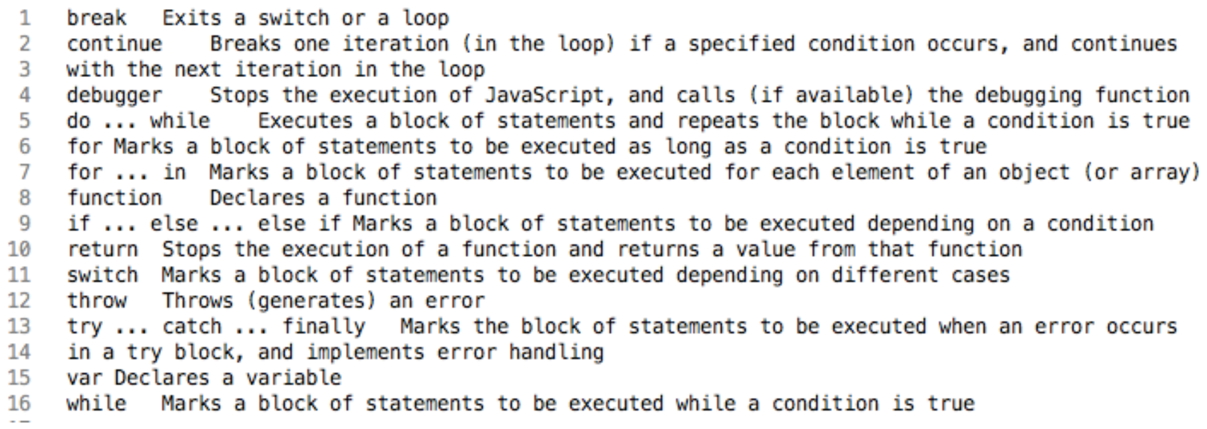
The entire library can be found at: https://github.com/mbostock/d3/wiki/API-Reference



```
d3.geom.voronoi - create a Voronoi layout with default accessors.
voronoi - compute the Voronoi tessellation for the specified points.
voronoi.x - get or set the x-coordinate accessor for each point.
voronoi.y - get or set the y-coordinate accessor for each point.
voronoi.clipExtent - get or set the clip extent for the tesselation.
voronoi.links - compute the Delaunay mesh as a network of links.
voronoi.triangles - compute the Delaunay mesh as a triangular tessellation.
d3.geom.quadtree - constructs a quadtree for an array of points.
quadtree.add - add a point to the quadtree.
quadtree.visit - recursively visit nodes in the quadtree.
quadtree.find - find the closest point in the quadtree.
d3.geom.polygon - create a polygon from the specified array of points.
polygon.area - compute the counterclockwise area of this polygon.
polygon.centroid - compute the area centroid of this polygon.
polygon.clip - clip the specified polygon to this polygon.
d3.geom.hull - create a convex hull layout with default accessors.
hull - compute the convex hull for the given array of points.
hull.x - get or set the x-coordinate accessor.
hull.y - get or set the y-coordinate accessor.
```

Appendix E Examples from the Dictionary for JavaScript

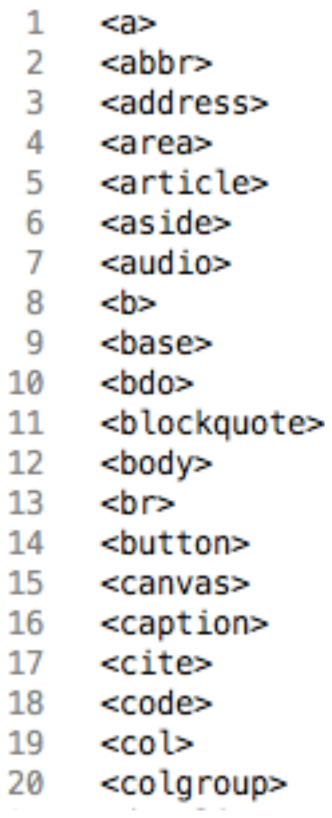
The entire library can be found at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference



```
break   Exits a switch or a loop
continue    Breaks one iteration (in the loop) if a specified condition occurs, and continues
with the next iteration in the loop
debugger    Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while    Executes a block of statements and repeats the block while a condition is true
for Marks a block of statements to be executed as long as a condition is true
for ... in  Marks a block of statements to be executed for each element of an object (or array)
function    Declares a function
if ... else ... else if Marks a block of statements to be executed depending on a condition
return  Stops the execution of a function and returns a value from that function
switch  Marks a block of statements to be executed depending on different cases
throw   Throws (generates) an error
try ... catch ... finally   Marks the block of statements to be executed when an error occurs
in a try block, and implements error handling
var Declares a variable
while   Marks a block of statements to be executed while a condition is true
```

Appendix F Examples from the Dictionary for HTML

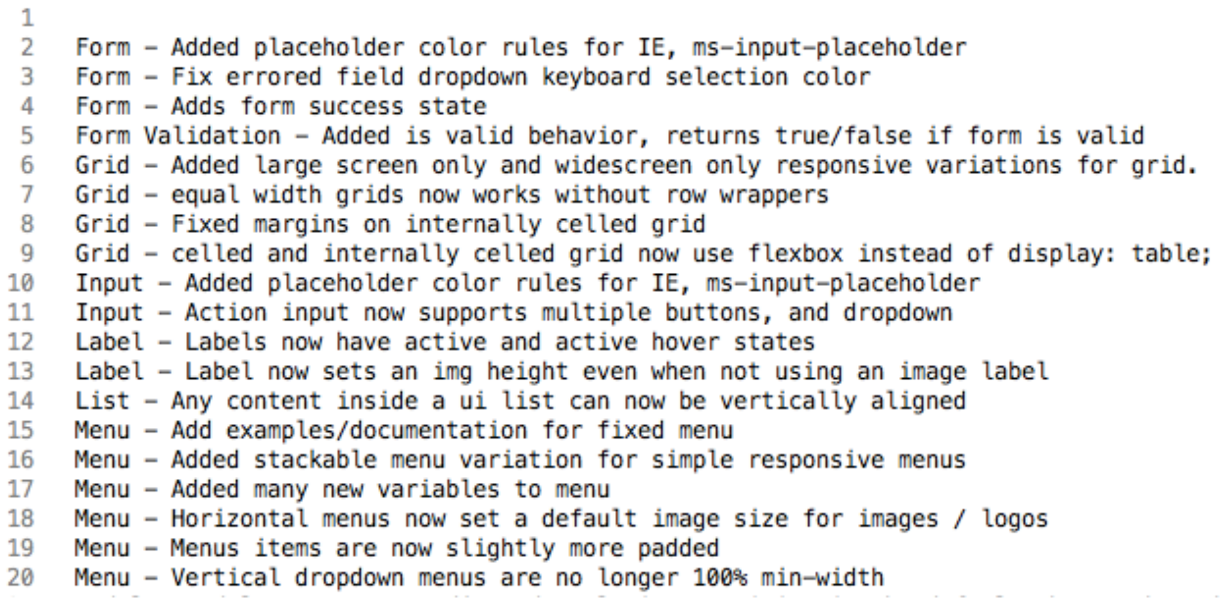
The entire library can be found at: http://www.w3schools.com/tags/



```
<a>
<abbr>
<address>
<area>
<article>
<aside>
<audio>
<b>
<base>
<bdo>
<blockquote>
<body>
<br>
<button>
<canvas>
<caption>
<cite>
<code>
<col>
<colgroup>
```

Appendix G Examples from the Dictionary for Semantic-ui

The entire library can be found at: http://semantic-ui.com/kitchen-sink.html#/modules

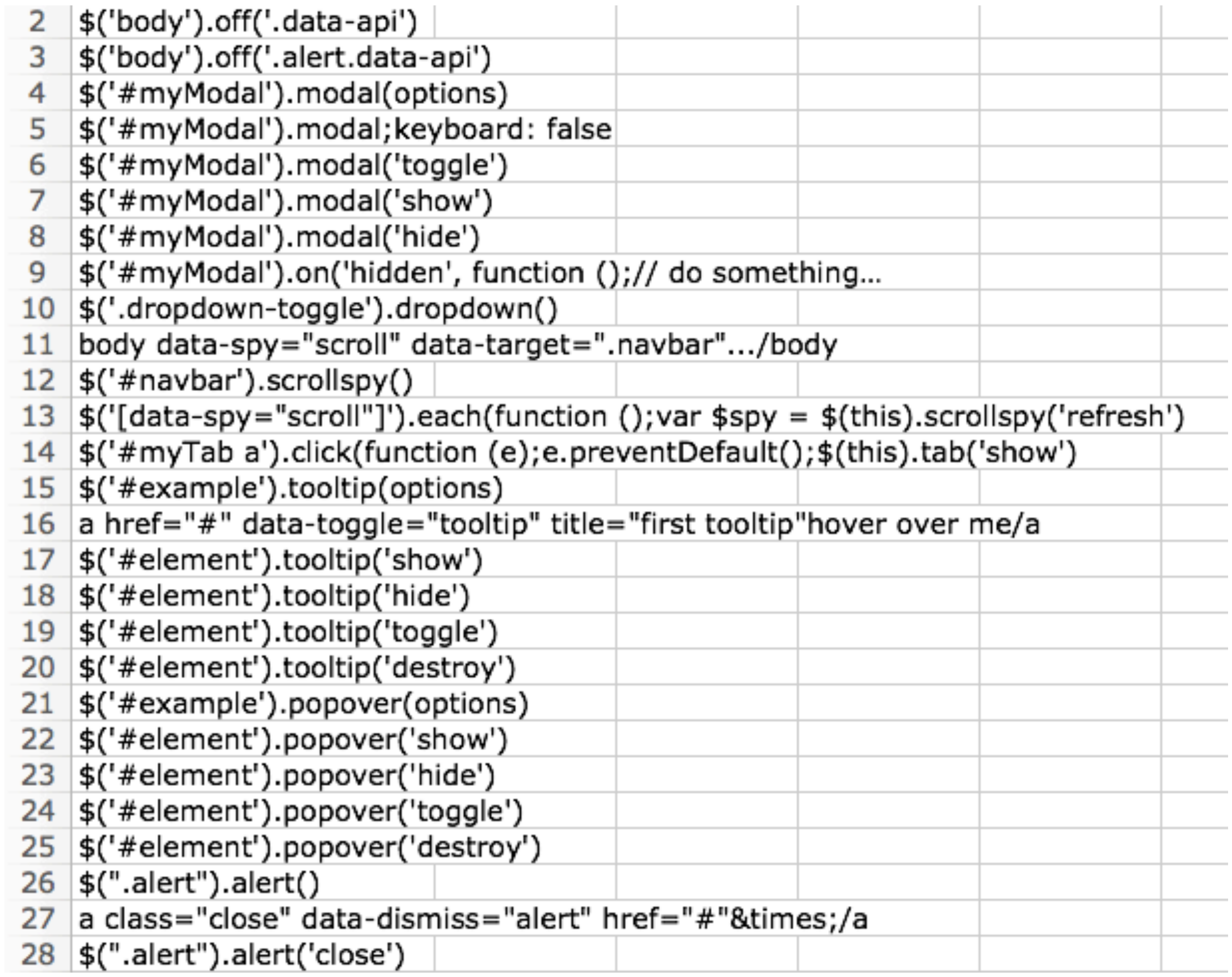


```

Form - Added placeholder color rules for IE, ms-input-placeholder
Form - Fix errored field dropdown keyboard selection color
Form - Adds form success state
Form Validation - Added is valid behavior, returns true/false if form is valid
Grid - Added large screen only and widescreen only responsive variations for grid.
Grid - equal width grids now works without row wrappers
Grid - Fixed margins on internally celled grid
Grid - celled and internally celled grid now use flexbox instead of display: table;
Input - Added placeholder color rules for IE, ms-input-placeholder
Input - Action input now supports multiple buttons, and dropdown
Label - Labels now have active and active hover states
Label - Label now sets an img height even when not using an image label
List - Any content inside a ui list can now be vertically aligned
Menu - Add examples/documentation for fixed menu
Menu - Added stackable menu variation for simple responsive menus
Menu - Added many new variables to menu
Menu - Horizontal menus now set a default image size for images / logos
Menu - Menus items are now slightly more padded
Menu - Vertical dropdown menus are no longer 100% min-width
```

Appendix H Examples from the Dictionary for BootstrapJS

The entire library can be found at: http://getbootstrap.com/javascript/



2	$('body').off('.data-api')
3	$('body').off('.alert.data-api')
4	$('#myModal').modal(options)
5	$('#myModal').modal;keyboard: false
6	$('#myModal').modal('toggle')
7	$('#myModal').modal('show')
8	$('#myModal').modal('hide')
9	$('#myModal').on('hidden', function ();// do something...
10	$('.dropdown-toggle').dropdown()
11	body data-spy="scroll" data-target=".navbar".../body
12	$('#navbar').scrollspy()
13	$('[data-spy="scroll"]').each(function ();var $spy = $(this).scrollspy('refresh')
14	$('#myTab a').click(function (e);e.preventDefault();$(this).tab('show')
15	$('#example').tooltip(options)
16	a href="#" data-toggle="tooltip" title="first tooltip"hover over me/a
17	$('#element').tooltip('show')
18	$('#element').tooltip('hide')
19	$('#element').tooltip('toggle')
20	$('#element').tooltip('destroy')
21	$('#example').popover(options)
22	$('#element').popover('show')
23	$('#element').popover('hide')
24	$('#element').popover('toggle')
25	$('#element').popover('destroy')
26	$(".alert").alert()
27	a class="close" data-dismiss="alert" href="#"&times;/a
28	$(".alert").alert('close')

Appendix I Function similarity for map components

	Hacker1	Hacker2	Hacker3	Hacker4	Hacker5	Hacker6	Hacker7	Hacker8	Hacker9	Hacker10	Hacker11	Hacker12	Hacker13
Hacker1	1.0000	0.1386	0.1808	0.1742	0.0615	0.0986	0.0694	0.3113	0.2208	0.1104	0.2159	0.0386	0.1709
Hacker2		1.0000	0.1643	0.0176	0.1194	0.0968	0.0796	0.0725	0.0899	0.1380	0.1149	0.1513	0.1000
Hacker3			1.0000	0.0309	0.0179	0.0919	0.1345	0.0250	0.0426	0.1503	0.0679	0.1079	0.0384
Hacker4				1.0000	0.0860	0.1328	0.1788	0.1730	0.1051	0.1929	0.0453	0.0078	0.0656
Hacker5					1.0000	0.0594	0.1561	0.0272	0.0968	0.1642	0.0110	0.2098	0.0695
Hacker6						1.0000	0.1481	0.1003	0.0262	0.0864	0.1033	0.1605	0.0840
Hacker7							1.0000	0.1214	0.2088	0.1417	0.2085	0.1062	0.1039
Hacker8								1.0000	0.1539	0.0330	0.1434	0.0942	0.1672
Hacker9									1.0000	0.0981	0.0973	0.1936	0.0580
Hacker10										1.0000	0.4768	0.1329	0.1740
Hacker11											1.0000	0.0427	0.2099
Hacker12												1.0000	0.1349
Hacker13													1.0000
Hacker14													
Hacker15													
Hacker16													
Hacker17													
Hacker18													
Hacker19													
Hacker20													
Hacker21													
Hacker22													
Hacker23													
Hacker24													
Hacker25													
Hacker26													

	Hacker14	Hacker15	Hacker16	Hacker17	Hacker18	Hacker19	Hacker20	Hacker21	Hacker22	Hacker23	Hacker24	Hacker25	Hacker26
Hacker1	0.1790	0.2106	0.1722	0.1728	0.1169	0.2266	0.0527	0.0818	0.1909	0.0093	0.2623	0.1760	0.0861
Hacker2	0.0562	0.0264	0.1164	0.0091	0.1241	0.0339	0.1130	0.1791	0.0248	0.0124	0.1781	0.0555	0.0856
Hacker3	0.0482	0.1453	0.2018	0.0465	0.1492	0.1210	0.0191	0.1620	0.1684	0.1676	0.1440	0.1799	0.2358
Hacker4	0.0038	0.1263	0.1201	0.1820	0.0547	0.1738	0.1349	0.1862	0.0752	0.1309	0.1246	0.0756	0.1238
Hacker5	0.1811	0.0346	0.1667	0.1212	0.1669	0.1559	0.0866	0.2138	0.1476	0.0427	0.0316	0.0807	0.1979
Hacker6	0.0906	0.0464	0.1741	0.1257	0.1094	0.0573	0.0941	0.2116	0.0918	0.0213	0.0999	0.1461	0.1011
Hacker7	0.1955	0.1846	0.0569	0.1413	0.1149	0.1304	0.1562	0.1182	0.1607	0.1106	0.0939	0.2573	0.1837
Hacker8	0.0452	0.1398	0.0355	0.1232	0.0625	0.1403	0.1576	0.1009	0.1617	0.0179	0.1309	0.1593	0.0132
Hacker9	0.2243	0.1434	0.1221	0.1393	0.0757	0.0977	0.1936	0.0802	0.1598	0.1339	0.0826	0.0544	0.0085
Hacker10	0.1856	0.1994	0.1258	0.1896	0.1738	0.1121	0.0422	0.0784	0.1609	0.0339	0.0599	0.0427	0.0528
Hacker11	0.0275	0.1856	0.1097	0.0282	0.0309	0.1835	0.0569	0.0101	0.0335	0.1986	0.0829	0.0210	0.0079
Hacker12	0.1835	0.0673	0.1560	0.1002	0.0361	0.1951	0.2159	0.1357	0.2007	0.1473	0.0493	0.0492	0.1482
Hacker13	0.0141	0.1537	0.0665	0.0530	0.0089	0.1929	0.1795	0.0424	0.1865	0.1176	0.0655	0.0816	0.0626
Hacker14	1.0000	0.1340	0.1537	0.0756	0.1066	0.1246	0.2119	0.0401	0.0639	0.1911	0.1661	0.0780	0.0785
Hacker15		1.0000	0.0483	0.1036	0.1441	0.0233	0.0839	0.1910	0.2771	0.1784	0.0779	0.1720	0.0226
Hacker16			1.0000	0.0443	0.0290	0.0473	0.1524	0.0158	0.1225	0.1963	0.1176	0.0904	0.1300
Hacker17				1.0000	0.0355	0.1053	0.1598	0.1662	0.1644	0.0625	0.1354	0.1472	0.0327
Hacker18					1.0000	0.1422	0.0674	0.1384	0.1018	0.0468	0.0947	0.2141	0.1260
Hacker19						1.0000	0.1265	0.0873	0.0453	0.1449	0.1418	0.1062	0.1123
Hacker20							1.0000	0.0581	0.0715	0.0735	0.2087	0.0354	0.0463
Hacker21								1.0000	0.1967	0.0868	0.2184	0.1224	0.1131
Hacker22									1.0000	0.0513	0.1314	0.1961	0.0446
Hacker23										1.0000	0.0314	0.2030	0.1925
Hacker24											1.0000	0.1974	0.0605
Hacker25												1.0000	0.2056
Hacker26													1.0000

Appendix J Function similarity for visualized graph components

	Hacker1	Hacker2	Hacker3	Hacker4	Hacker5	Hacker6	Hacker7	Hacker8	Hacker9	Hacker10	Hacker11	Hacker12	Hacker13
Hacker1	1.0000	0.0383	0.0769	0.0783	0.0701	0.0582	0.1153	0.2577	0.1282	0.1926	0.0725	0.1248	0.0144
Hacker2		1.0000	0.0247	0.0856	0.0108	0.0483	0.0697	0.1071	0.0755	0.1991	0.0666	0.1574	0.1067
Hacker3			1.0000	0.1439	0.0167	0.2174	0.1832	0.0431	0.1031	0.0831	0.2023	0.1279	0.0326
Hacker4				1.0000	0.2196	0.0234	0.1936	0.2017	0.1745	0.0656	0.1397	0.2098	0.0338
Hacker5					1.0000	0.0931	0.1121	0.0892	0.0897	0.1977	0.0514	0.1985	0.0635
Hacker6						1.0000	0.0955	0.0362	0.0550	0.0536	0.0046	0.2113	0.2038
Hacker7							1.0000	0.1091	0.0356	0.0669	0.0412	0.0063	0.0819
Hacker8								1.0000	0.0560	0.0625	0.0856	0.1774	0.0263
Hacker9									1.0000	0.0133	0.0814	0.0825	0.0349
Hacker10										1.0000	0.2343	0.0950	0.0742
Hacker11											1.0000	0.0390	0.0159
Hacker12												1.0000	0.1113
Hacker13													1.0000
Hacker14													
Hacker15													
Hacker16													
Hacker17													
Hacker18													
Hacker19													
Hacker20													
Hacker21													
Hacker22													
Hacker23													
Hacker24													
Hacker25													
Hacker26													

	Hacker14	Hacker15	Hacker16	Hacker17	Hacker18	Hacker19	Hacker20	Hacker21	Hacker22	Hacker23	Hacker24	Hacker25	Hacker26
Hacker1	0.1378	0.1767	0.1296	0.0973	0.0882	0.2368	0.0568	0.0910	0.1482	0.0528	0.2316	0.1308	0.1035
Hacker2	0.0311	0.0684	0.2155	0.0713	0.0044	0.0737	0.0430	0.1321	0.0696	0.0701	0.0667	0.0984	0.0914
Hacker3	0.0673	0.1038	0.1963	0.0485	0.0753	0.0540	0.0930	0.1170	0.0806	0.1057	0.1345	0.1217	0.2298
Hacker4	0.0151	0.1237	0.1640	0.2094	0.0071	0.0592	0.2146	0.1854	0.0141	0.0766	0.1319	0.1703	0.0953
Hacker5	0.1207	0.0081	0.1816	0.1050	0.0863	0.0881	0.2031	0.1020	0.0932	0.0816	0.1103	0.1005	0.1834
Hacker6	0.0857	0.0256	0.0282	0.0840	0.0190	0.0025	0.0665	0.1432	0.0166	0.0537	0.1741	0.0960	0.0917
Hacker7	0.0228	0.0894	0.0143	0.0162	0.0622	0.0581	0.0961	0.1155	0.0532	0.0283	0.1082	0.1049	0.1593
Hacker8	0.0386	0.0078	0.2022	0.0784	0.1162	0.0988	0.1054	0.1118	0.0734	0.0501	0.2186	0.1167	0.0674
Hacker9	0.0726	0.1179	0.1709	0.0715	0.0929	0.0942	0.1199	0.0461	0.0505	0.0204	0.0533	0.0685	0.0463
Hacker10	0.0133	0.0082	0.0283	0.0704	0.0490	0.2060	0.0366	0.1499	0.0507	0.1945	0.0379	0.0310	0.1182
Hacker11	0.0930	0.1329	0.0485	0.2094	0.1646	0.1544	0.1192	0.2188	0.0472	0.0555	0.0639	0.0437	0.2127
Hacker12	0.0193	0.1912	0.0526	0.0156	0.0327	0.2073	0.1169	0.0693	0.1305	0.1106	0.0388	0.0172	0.1837
Hacker13	0.0712	0.0985	0.0984	0.0987	0.0909	0.2334	0.1075	0.0707	0.1284	0.0706	0.0374	0.1077	0.0118
Hacker14	1.0000	0.1165	0.1394	0.1059	0.0586	0.0821	0.0886	0.0523	0.1029	0.1136	0.1437	0.0560	0.0126
Hacker15		1.0000	0.2021	0.1041	0.0955	0.0265	0.0881	0.2097	0.2093	0.1499	0.0313	0.0985	0.0213
Hacker16			1.0000	0.0527	0.0019	0.1647	0.1893	0.0570	0.0330	0.0919	0.1433	0.2145	0.0696
Hacker17				1.0000	0.0441	0.0730	0.0092	0.0571	0.0308	0.0831	0.0967	0.1297	0.0296
Hacker18					1.0000	0.0065	0.1828	0.1164	0.0029	0.1443	0.1071	0.1475	0.0723
Hacker19						1.0000	0.1555	0.0978	0.1216	0.0899	0.0982	0.0829	0.1578
Hacker20							1.0000	0.1757	0.1066	0.0685	0.1510	0.0454	0.1525
Hacker21								1.0000	0.2109	0.0713	0.0865	0.0506	0.0534
Hacker22									1.0000	0.0776	0.0942	0.1273	0.0890
Hacker23										1.0000	0.0816	0.0866	0.0987
Hacker24											1.0000	0.1700	0.0901
Hacker25												1.0000	0.1437
Hacker26													1.0000

Appendix K Function similarity for interaction form components

	Hacker1	Hacker2	Hacker3	Hacker4	Hacker5	Hacker6	Hacker7	Hacker8	Hacker9	Hacker10	Hacker11	Hacker12	Hacker13
Hacker1	1.0000	0.1030	0.1540	0.0969	0.2357	0.1293	0.1572	0.2717	0.3416	0.1736	0.2058	0.0720	0.0479
Hacker2		1.0000	0.1766	0.3173	0.1535	0.2365	0.2525	0.2521	0.1445	0.2533	0.0954	0.3020	0.1201
Hacker3			1.0000	0.1093	0.1431	0.1097	0.2776	0.0910	0.0667	0.0189	0.1577	0.0698	0.1524
Hacker4				1.0000	0.1733	0.1064	0.2782	0.2449	0.0821	0.2599	0.0257	0.1582	0.1684
Hacker5					1.0000	0.0445	0.1326	0.0778	0.1227	0.0911	0.2747	0.2153	0.2438
Hacker6						1.0000	0.1872	0.0756	0.1764	0.2439	0.2447	0.3034	0.1596
Hacker7							1.0000	0.1508	0.1202	0.3274	0.2122	0.0110	0.1714
Hacker8								1.0000	0.1015	0.0908	0.1393	0.0730	0.0532
Hacker9									1.0000	0.1017	0.0801	0.0649	0.0385
Hacker10										1.0000	0.2635	0.2916	0.1434
Hacker11											1.0000	0.1388	0.2527
Hacker12												1.0000	0.2943
Hacker13													1.0000
Hacker14													
Hacker15													
Hacker16													
Hacker17													
Hacker18													
Hacker19													
Hacker20													
Hacker21													
Hacker22													
Hacker23													
Hacker24													
Hacker25													
Hacker26													

	Hacker14	Hacker15	Hacker16	Hacker17	Hacker18	Hacker19	Hacker20	Hacker21	Hacker22	Hacker23	Hacker24	Hacker25	Hacker26
Hacker1	0.0692	0.2995	0.0741	0.1855	0.1928	0.1729	0.2540	0.2117	0.1882	0.2496	0.2940	0.2669	0.1142
Hacker2	0.1667	0.1837	0.1945	0.0128	0.1717	0.1380	0.1196	0.0284	0.0962	0.1672	0.1443	0.1651	0.0861
Hacker3	0.2672	0.1052	0.3143	0.0766	0.1402	0.0989	0.1767	0.0127	0.2482	0.0000	0.3147	0.1967	0.2750
Hacker4	0.0734	0.1131	0.0283	0.0669	0.2696	0.0334	0.2443	0.1998	0.3232	0.3245	0.3191	0.1354	0.1712
Hacker5	0.0957	0.0789	0.0012	0.0168	0.1254	0.1753	0.2426	0.2733	0.0279	0.1975	0.0062	0.2775	0.2257
Hacker6	0.1083	0.3275	0.0056	0.0449	0.2814	0.2075	0.0308	0.2540	0.1452	0.0481	0.2998	0.1519	0.1140
Hacker7	0.2255	0.0315	0.2799	0.2303	0.2285	0.1029	0.0525	0.1007	0.1258	0.2785	0.0402	0.2713	0.1570
Hacker8	0.0638	0.1098	0.1899	0.0482	0.1148	0.1278	0.2226	0.3285	0.0318	0.0539	0.2892	0.2253	0.2388
Hacker9	0.1303	0.1272	0.0132	0.0068	0.0334	0.1697	0.2563	0.2567	0.2447	0.0447	0.1286	0.2724	0.1882
Hacker10	0.1891	0.3073	0.1738	0.2914	0.0754	0.1550	0.2488	0.0685	0.1310	0.0487	0.0245	0.1882	0.2847
Hacker11	0.1248	0.0507	0.1602	0.2560	0.0901	0.2371	0.0927	0.0411	0.1794	0.3128	0.0529	0.0228	0.1235
Hacker12	0.2329	0.0977	0.2374	0.1692	0.1993	0.2572	0.0236	0.2878	0.0442	0.3101	0.1559	0.1056	0.2038
Hacker13	0.1886	0.1857	0.3083	0.1198	0.1519	0.2167	0.2321	0.3220	0.1510	0.0495	0.2465	0.0596	0.1499
Hacker14	1.0000	0.3154	0.2412	0.2272	0.1074	0.1913	0.1957	0.1418	0.2491	0.2062	0.2490	0.1901	0.2654
Hacker15		1.0000	0.0491	0.2618	0.2162	0.1087	0.1422	0.1427	0.3085	0.0610	0.0255	0.0354	0.1454
Hacker16			1.0000	0.3157	0.0204	0.1530	0.2677	0.0112	0.2277	0.2947	0.1866	0.2572	0.3094
Hacker17				1.0000	0.1668	0.0791	0.2280	0.0397	0.1697	0.1372	0.0200	0.3305	0.0059
Hacker18					1.0000	0.0302	0.1680	0.0912	0.0527	0.1900	0.1067	0.1417	0.0229
Hacker19						1.0000	0.0281	0.2716	0.0377	0.1450	0.0744	0.0723	0.0376
Hacker20							1.0000	0.1950	0.0973	0.1328	0.1645	0.2110	0.1014
Hacker21								1.0000	0.1140	0.0788	0.1609	0.2040	0.0942
Hacker22									1.0000	0.1220	0.0940	0.2426	0.0907
Hacker23										1.0000	0.2997	0.2465	0.0389
Hacker24											1.0000	0.1054	0.2032
Hacker25												1.0000	0.1892
Hacker26													1.0000

Appendix L Overall function similarity

	Hacker1	Hacker2	Hacker3	Hacker4	Hacker5	Hacker6	Hacker7	Hacker8	Hacker9	Hacker10	Hacker11	Hacker12	Hacker13
Hacker1	1.0000	0.0933	0.1373	0.1165	0.1224	0.0954	0.1139	0.2802	0.2302	0.1589	0.1647	0.0785	0.0777
Hacker2		1.0000	0.1219	0.1402	0.0946	0.1272	0.1339	0.1439	0.1033	0.1968	0.0923	0.2035	0.1090
Hacker3			1.0000	0.0947	0.0592	0.1397	0.1984	0.0530	0.0708	0.0841	0.1426	0.1019	0.0745
Hacker4				1.0000	0.1596	0.0875	0.2168	0.2065	0.1206	0.1728	0.0702	0.1252	0.0892
Hacker5					1.0000	0.0657	0.1336	0.0647	0.1031	0.1510	0.1124	0.2079	0.1256
Hacker6						1.0000	0.1436	0.0707	0.0858	0.1280	0.1175	0.2250	0.1491
Hacker7							1.0000	0.1271	0.1215	0.1787	0.1540	0.0412	0.1191
Hacker8								1.0000	0.1038	0.0621	0.1228	0.1149	0.0823
Hacker9									1.0000	0.0710	0.0863	0.1137	0.0438
Hacker10										1.0000	0.3249	0.1732	0.1305
Hacker11											1.0000	0.0735	0.1595
Hacker12												1.0000	0.1801
Hacker13													1.0000
Hacker14													
Hacker15													
Hacker16													
Hacker17													
Hacker18													
Hacker19													
Hacker20													
Hacker21													
Hacker22													
Hacker23													
Hacker24													
Hacker25													
Hacker26													

	Hacker14	Hacker15	Hacker16	Hacker17	Hacker18	Hacker19	Hacker20	Hacker21	Hacker22	Hacker23	Hacker24	Hacker25	Hacker26
Hacker1	0.1287	0.2289	0.1253	0.1519	0.1326	0.2121	0.1212	0.1282	0.1758	0.1039	0.2626	0.1912	0.1013
Hacker2	0.0846	0.0929	0.1755	0.0311	0.1001	0.0819	0.0919	0.1132	0.0635	0.0832	0.1297	0.1064	0.0877
Hacker3	0.1275	0.1181	0.2375	0.0572	0.1216	0.0913	0.0963	0.0972	0.1657	0.0911	0.1977	0.1661	0.2469
Hacker4	0.0308	0.1210	0.1042	0.1528	0.1105	0.0888	0.1979	0.1904	0.1375	0.1773	0.1919	0.1271	0.1301
Hacker5	0.1325	0.0405	0.1165	0.0810	0.1262	0.1398	0.1774	0.1964	0.0896	0.1073	0.0494	0.1529	0.2024
Hacker6	0.0949	0.1332	0.0693	0.0849	0.1366	0.0891	0.0638	0.2030	0.0845	0.0410	0.1913	0.1313	0.1023
Hacker7	0.1479	0.1019	0.1170	0.1293	0.1352	0.0972	0.1016	0.1114	0.1132	0.1392	0.0808	0.2111	0.1667
Hacker8	0.0492	0.0858	0.1426	0.0833	0.0979	0.1223	0.1619	0.1804	0.0890	0.0406	0.2129	0.1671	0.1065
Hacker9	0.1424	0.1295	0.1021	0.0725	0.0673	0.1206	0.1899	0.1277	0.1517	0.0663	0.0882	0.1318	0.0810
Hacker10	0.1294	0.1716	0.1093	0.1838	0.0994	0.1577	0.1092	0.0989	0.1142	0.0924	0.0408	0.0873	0.1519
Hacker11	0.0818	0.1231	0.1062	0.1645	0.0952	0.1916	0.0896	0.0900	0.0867	0.1890	0.0666	0.0292	0.1147
Hacker12	0.1453	0.1188	0.1487	0.0950	0.0894	0.2199	0.1188	0.1643	0.1251	0.1894	0.0814	0.0573	0.1786
Hacker13	0.0913	0.1460	0.1577	0.0905	0.0839	0.2143	0.1730	0.1450	0.1553	0.0792	0.1165	0.0830	0.0748
Hacker14	1.0000	0.1886	0.1781	0.1362	0.0909	0.1327	0.1654	0.0780	0.1387	0.1703	0.1862	0.1080	0.1188
Hacker15		1.0000	0.0999	0.1565	0.1519	0.0528	0.1047	0.1811	0.2650	0.1298	0.0449	0.1020	0.0631
Hacker16			1.0000	0.1376	0.0171	0.1217	0.2031	0.0280	0.1278	0.1943	0.1492	0.1874	0.1697
Hacker17				1.0000	0.0821	0.0858	0.1323	0.0877	0.1217	0.0943	0.0840	0.2025	0.0227
Hacker18					1.0000	0.0596	0.1394	0.1153	0.0524	0.1270	0.1028	0.1678	0.0737
Hacker19						1.0000	0.1034	0.1522	0.0682	0.1266	0.1048	0.0871	0.1026
Hacker20							1.0000	0.1429	0.0918	0.0916	0.1747	0.0973	0.1001
Hacker21								1.0000	0.1738	0.0790	0.1553	0.1257	0.0869
Hacker22									1.0000	0.0836	0.1065	0.1887	0.0748
Hacker23										1.0000	0.1376	0.1787	0.1101
Hacker24											1.0000	0.1576	0.1179
Hacker25												1.0000	0.1795
Hacker26													1.0000