

Copyright
by
Yamini Gotimukul
2014

The Thesis committee for Yamini Gotimukul
certifies that this is the approved version of the following thesis:

Visual Exploration Support for Cross-Project Porting

APPROVED BY

SUPERVISING COMMITTEE:

Christine L Julien, Supervisor

Miryung Kim, Co-Supervisor

Visual Exploration Support for Cross-Project Porting

by

Yamini Gotimukul, Bach. of Eng.

THESIS

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2014

Dedicated to my loving husband Guru Prasad Karanam.

Acknowledgments

I would like to express my gratitude to Dr. Miryung Kim for her guidance and continuous support throughout this thesis. I thank her for giving me an excellent research opportunity, for the critical reviews and the feedbacks. I would also like to thank Dr. Christine Julien for taking the time to read and for the constructive feedback on this thesis.

I am very grateful to my parents, Srinivas Gotimukul and Uma Devi Gotimukul for their love and enormous encouragement throughout my life. I thank my sister, Ashwini Gotimukul for always being there for me. I thank my in-laws, Narayana Murthy Karanam and Annapurna Karanam for their love and support. I appreciate the encouragement from Praveena Karanam and Rahul Mathukumalli. Finally, I would like to express my gratitude to my husband Guru Karanam for being a rock support to me throughout this journey.

Visual Exploration Support for Cross-Project Porting

Yamini Gotimukul, M.S.E
The University of Texas at Austin, 2014

Supervisors: Christine L Julien
Miryung Kim

Maintaining multiple variants of software systems is extremely difficult because developers often port edits and bug fixes during software evolution. This challenge particularly applies to closely related families of open source projects, such as BSD projects (FreeBSD, NetBSD and OpenBSD) with extensive cross-project porting activities. Developers encounter increasing obstacles in maintaining projects, particularly because of the difficulty in understanding historical artifacts involved in cross-system porting. Maintainers face the primary challenge of keeping track of the sources of ported edits, as it can be extremely time-consuming to mine historical data and track the source and target of patches. In the worst-case scenario, the maintainer has to mine through all historical data to ascertain the sources of ported code. Although current version control systems like CVS and GIT preserve historical data, the developer cannot easily identify and understand cross-system porting activities.

In this thesis, we address the aforementioned issues by designing and implementing software visualization support to analyze the long chain of cross-project porting activities for Open Source Softwares (OSS) and particularly for three BSD projects (FreeBSD, OpenBSD and NetBSD). We take into account the geographically distributed community of OSS developers and maintainers, hosting the visualization of the activities as a web application. This study aims to analyze the effects of visualization on cross-project porting activity awareness. To meet the study’s objective, we developed a web-based awareness tool, VIGNETTE, based on the results of REPERTOIRE [18] (which identifies the cross-project porting activities in BSD projects using release history).

This study focuses on two research questions: (1) How can visualization help novice open-source developers and maintainers gain insights into cross-project (projects evolving from the same code base) porting activities? (2) How can the visualization show the following: (a) a file-level association between peer projects (porting activities in cross-project files with similar file names), (b) the pairwise frequency of porting (the porting activity count between two cross-projects in a year), (c) the patch-file association (same patch id applied to different cross-project files), and (d) the developer to developer association based on cross-project porting activities (number of times the cross-project developers was involved in a common porting activity)? We conducted a user study with graduate students in the role of novice open-source developers interested in learning about cross-project porting activities. The results of the initial study showed that VIGNETTE could be very useful in answering the

questions about cross-project porting and in determining who was involved in a particular porting activity and when.

Table of Contents

Acknowledgments	v
Abstract	vi
List of Tables	xiii
List of Figures	xiv
Chapter 1. Introduction	1
1.1 Background	2
1.2 Visualizations	3
1.3 Research Problem	4
1.4 Contributions	6
1.5 Thesis Outline	7
Chapter 2. Literature Review	8
2.1 Analysis of Cross-Project Porting	8
2.2 Clone Detection Techniques	9
2.3 Tool Support and Mining Software Repositories for Cross-Project Activities	11
2.4 Visualization Aid for Cross-Project Activities	12
2.5 Web-based Tools for Analyzing Software Evolution	13
Chapter 3. Tree Visualization	14
3.1 Background	14
3.1.1 Motivation	15
3.1.2 A Hierarchical Representation	15
3.1.3 Comparison View	17
3.2 Coloring Scheme	18

3.3	Data Formatting	19
3.3.1	Sample Data with Example	20
3.4	Interactive Features	20
3.4.1	Graph Zoom In and Zoom Out	21
3.4.2	Graph Panning	21
3.4.3	Click Event	21
3.4.4	Additional Information Page	21
3.4.5	Learn More Page	22
3.5	Intended Benefits	22
Chapter 4. Bubble Chart Visualization		25
4.1	Background	25
4.1.1	Motivation	25
4.2	A Bubbled Representation	26
4.3	Coloring Scheme	27
4.4	Data Formatting	27
4.4.1	Sample Data with Example	28
4.5	Interactive Features	28
4.5.1	Bubble Mouseover	29
4.5.2	On Bubble Click	29
4.5.3	Additional Information Page	30
4.5.4	Learn More Page	30
4.6	Intended Benefits	31
Chapter 5. Developer Dependency Wheel Visualization		33
5.1	Background	33
5.1.1	Motivation	34
5.2	Developer Dependency Wheel	34
5.2.1	Learn More Page	36
5.3	Color Scheme	37
5.4	Data Formatting	37
5.5	Interactive Features	38
5.5.1	Mouseover	39
5.6	Intended Benefits	40

Chapter 6. Bipartite Visualization	41
6.1 Background	41
6.1.1 Motivation	41
6.2 Bipartite View	42
6.2.1 Comparison View	42
6.2.2 Learn More Page	44
6.3 Data Formatting	44
6.4 Interactive Features	46
6.4.1 Mouse Over	46
6.4.2 Mouse Click	46
6.5 Intended Benefits	47
Chapter 7. Implementation	48
7.1 Web Pages	48
7.2 APIs and Server	49
Chapter 8. Evaluation and Results	52
8.1 User Study Design	52
8.2 Study Experience	52
8.2.1 User Profiles	53
8.3 Results	54
8.3.1 Interpretation	54
8.3.2 Result Metrics	55
8.3.3 Research Questions	59
Chapter 9. Conclusion and Future Work	66
9.1 Threats To Validity	66
9.1.1 Construct Validity	66
9.1.2 Internal validity	67
9.1.3 External Validity	67
9.1.4 Conclusion Validity	67
9.2 Summary	68
9.3 Future Work	68

Appendices	69
Appendix A. User Study Reference Manual	70
A.1 Introduction	70
A.1.1 Web-based Visualization of Cross-Project Porting	71
A.1.2 Study Overview	72
A.1.3 Demo Agenda	72
A.1.4 Tasks	73
A.1.5 Feedback Questionnaire	86
Bibliography	89
Vita	94

List of Tables

3.1	Tree visualization coloring scheme interpretation	18
4.1	Bubble visualization coloring scheme interpretation	28
8.1	Tree visualization results summary	55
8.2	Bubble visualization results summary	56
8.3	Bipartite visualization results summary	57
8.4	Developer dependency visualization results summary	58
8.5	Overall feedback results summary	59
8.6	User background summary	60

List of Figures

1.1	Thumbnail view of four visualizations	4
3.1	FreeBSD hierarchical tree view	16
3.2	Comparison views of FreeBSD and NetBSD projects	17
3.3	Pop-up dialogue box displayed when a node is clicked	22
3.4	Tree visualization additional information page	23
3.5	Learn more button	23
3.6	Tree visualization learn more page	24
4.1	Porting frequency in bubble visualization	26
4.2	Bubble mouse over event	29
4.3	Bubble mouse click event	30
4.4	Bubble visualization additional information page	31
4.5	Learn more button	31
4.6	Bubble visualization learn more page	32
5.1	Developer dependency visualization	35
5.2	Developer dependency wheel web page	36
5.3	Developer dependency page showing back button	36
5.4	Learn more page	37
5.5	Dependency matrix for 50 FreeBSD and NetBSD developers	38
5.6	Mouse over action on wheel arc	39
5.7	Mouse over action on wheel chord	39
6.1	Bipartite comparison view	43
6.2	Bipartite visualization web page	44
6.3	Learn more page	45
6.4	Bipartite visualization input data format	45
6.5	Bipartite chart mouseover on the patch ID	46

6.6	Bipartite view click operations	47
7.1	Common navigation and page ribbon	49
7.2	Thumbnail views of visualizations	49
7.3	Learn more page	50
8.1	Tree visualization results	58
8.2	Bubble visualization results	60
8.3	Bipartite visualization results	61
8.4	Developer dependency visualization results	61
8.5	Overall results for VIGNETTE	62
8.6	Summary of normalized scores	62

Chapter 1

Introduction

Open Source Softwares (OSS) often needs to maintain different variants of the same project. These variants exist for several reasons, such as different project goals, personality clashes [26] or even licensing restrictions [8]. For example, one of the main reasons for the BSD project to maintain three variants (FreeBSD, NetBSD and OpenBSD) is because of different project goals. This common practice of creating different variants of the same project (*software forking*) introduces complexity to the process of software maintenance. Software forking activity spawns “competing” projects that cannot later exchange code, thus splitting the potential developer community [20]. To maintain forked projects, developers need to port code from peer projects, which involves manual adaptation of features and bug fixes (a code exchange activity). Moreover, developers may find it difficult to learn about patch updates (for example, updates made to a previously applied bug fix patch) and porting sources across peer projects because of the split in the developers’ community. To overcome the difficulties associated with learning about porting activities by mining the code repository or from discussion forums, developers may require a visualization system that summarizes these activities.

1.1 Background

Many forked open-source projects require a significant amount of porting activities in the maintenance phase. For example, forked BSD projects (FreeBSD, NetBSD and OpenBSD) evolve from the same code base and may need the same bug fix patch to applied to all three projects (Porting). An in-depth case study conducted on BSD projects using 18 years of release history reveals that porting is an important regularly-occurring activity, and that ported code is less defect-prone [18]. The importance of porting activities cannot be overstated; for example, Unix systems lost market share compared to Windows mainly due to the porting restriction imposed on a large number of forked Unix projects [26]. (These restrictions on forked projects resulted from *proprietary innovations* that caused the forked Unix OSs to differ from each other while restricting access to the innovations made in source code [27].) Unlike Unix OS, Linux OS supports software porting across different forked projects (cross-project), allowing the innovations to be available to various user communities [27]. Clearly, porting can have a huge impact on the project success. REPERTOIRE represents the first work to successfully identify cross-project porting activities in BSD projects [18]. Based on the results of REPERTOIRE [18], this thesis proposes a simple solution that visualizes cross-system porting activities using a web-based application. This application attempts to ease the developers' burden in performing investigation of cross-project porting.

1.2 Visualizations

We developed a web-based awareness tool, VIGNETTE. Because OSS developer communities are spread across the globe, we took this geographical distribution into consideration by making the tool web accessible. VIGNETTE currently analyzes the porting activities for BSD peer projects. As input, the tool takes the porting activity data identified by REPERTOIRE [18] and visualizes the results in different views. VIGNETTE provides four visualizations:

1. **Tree** visualization shows in hierarchy the porting activities based on the file level similarity across peer projects.
2. **Bubble** visualization shows the pairwise porting frequency as a bubble chart.
3. **Bipartite** visualization shows a bipartite chart of the porting activities based on the same patch IDs applied to the same project and peer projects files.
4. **Developer Dependency** visualization represents as a dependency wheel the number of times the developer pair was involved in a cross-project porting activity.

Figure 1.1 displays the thumbnail view of the four visualizations as displayed on the Welcome page of VIGNETTE.

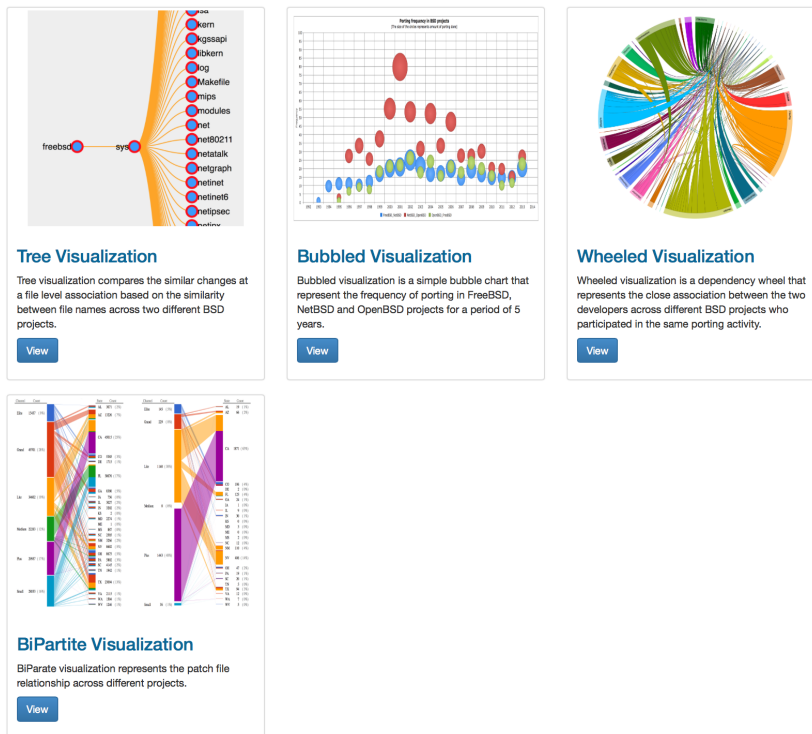


Figure 1.1: Thumbnail view of four visualizations

1.3 Research Problem

The main purpose of this thesis is to investigate how visualization of cross-system porting can help developers understand who ported what and when. This study hypothesizes that understanding the cross-project dependencies and porting activities are key aspects for successful maintenance of OSS projects.

The following summarizes the research questions and study results:

1. How can the visualizations help novice open-source developers

and maintainers gain insights into cross-project porting activities?

This research question seeks to investigate if visualization could help OSS developers learn about cross-project porting activities; it also seeks to identify how quickly they could learn about these activities. In the study, both users identified the porting activities correctly, completing the identifications tasks within 20 minutes.

2. How well can VIGNETTE show the following aspects of cross-project porting activities?

(a) A file level association.

We investigated if tree visualization could help developers identify porting activities in similar files between FreeBSD and NetBSD.

(b) The pairwise frequency of porting.

We investigated if bubble visualization could help the OSS community learn about porting frequency among project pairs over the entire project history.

(c) Patch-file association.

We investigated if bipartite visualization could help novice developers learn about patch-file associations (association of the same patch Id with different files in peer projects) among the FreeBSD project patches and files in the FreeBSD and NetBSD projects.

(d) **The developer to developer association.**

Using this question, we investigated if the developer dependency visualization could help novice developers learn about the developer-to-developer (number of times the developer pair was involved in a porting activity) association.

The initial user study was conducted with two participants. In the study, both participants could perform the tasks correctly. Both users gave a higher rating (mostly 4 or 5 out of 5) for VIGNETTE. The following user comments demonstrate users' satisfaction, as collected during each study session:

“I think the bipartite visualization is the most useful.”

“Visualization can represent the raw data very well.”

1.4 Contributions

The main contributions of this thesis include the following. We have analyzed the raw porting data identified by REPERTOIRE [18] in four different aspects and visually presented the data. In addition, we have developed a novel web-based tool, VIGNETTE, to make porting analysis results easily accessible to any developer from anywhere in the world.

1.5 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 summarizes the literature regarding prior works on this topic. Chapter 3 describes various aspects of the tree visualization, while Chapter 4 presents the features of the bubble chart visualization. Chapter 5 provides details into the bipartite visualization, and Chapter 6 discusses the developer dependency wheel visualization. Chapter 7 discusses the web-based tool implementation. Chapter 8 provides the evaluation methodology and summarizes the study results. Finally, Chapter 9 concludes the thesis by summarizing and discussing the study's limitations and by making recommendations for future work.

Chapter 2

Literature Review

This chapter summarizes various prior researches in closely related areas. The review on earlier works is presented in five categories: (1) cross-project porting activities, (2) clone detection techniques, (3) tool support and mining software repositories for cross-project activities, (4) visualization aid for cross-project porting activities, and (5) web-based tools for analyzing software repositories. The rest of the chapter is organized into five subsections (subsections 2.1 through 2.5, one for each of the categories listed above) that explain the findings, the limitations, and the results in comparison to this thesis.

2.1 Analysis of Cross-Project Porting

In an open source development context, developers often adapt code patches from similar projects. Ray et al. studied the cross-project porting activities in BSD peer projects using REPERTOIRE [18]. Ray et al. found that cross-system porting happens more often. They showed that porting rate does not decrease over time, and a significant portion of active committers port changes. Although REPERTOIRE [18] was a major milestone in this area,

the tool was designed to identify cross-system porting activities but lacked an intuitive method to present cross-project porting activity data to BSD developers. Tilbrook et al. studied the criteria for an effective porting strategy [23] when retargeting large-scale software to support new platforms. They showed that the methodology employed to specify the characteristics of target platforms is important for any porting strategy. German et al. identified that licensing issues between variants of related projects contributed to creation of different products, code siblings [11]. They showed that cross-project code migration happens from less restrictive licenses to more restrictive licenses.

None of the prior studies could visualize porting activities that might save OSS developers or contributors a great deal of time. In contrast, this work focuses on presenting porting activity data to developers so as to save their time in investigating cross-system porting.

2.2 Clone Detection Techniques

Porting is an activity of applying similar patches across variant projects. Therefore, clone detection and porting analysis are closely related. Kim et al. analyzed and presented evolutionary code clones, designing SoftGUESS [1] to visualize code clone data. Although SoftGUESS intuitively represents the code clones in various browser views, it is limited to present the clone visualizations in a desktop application. Krinke et al. proposed an approach that detects similar but not identical clones using program dependency graphs (PDG) [14], while Taires et al. developed an eclipse plug-in, CloneDR [22], to visualize the

clones. Rieger et al. created DUPLOC [21], a matrix-based clickable clone detector and a visualizer explorer. DUPLOC could detect and summarize clones at source code granularity in a report. Kapser et al. created CLICS (Clone Interpretation and Navigation System) [12] in order to understand the code cloning process. They showed that a sub-project constituting approximately 17% of the project's code held about 39% of the clones. Li et al. developed CP-Miner [16] to detect copy-pasted code and defects. They showed that CP-Miner could detect numerous clones as well as previously unidentified defects in Linux and FreeBSD. Balint et al. identified the patterns between developers involved in code copying [2]. They associated developers if they cloned the code at the same time in a similar notion of our developer-to-developer association based on porting activities. Koschke et al. presented a complete clone analysis, including identification of cloning root cause, detection, visualization, and removal of code clones [13]. Fisher et al. reported the code commonalities for three BSD projects [10].

It should be noted that none of these works have visualized the clones detected using methods similar to those presented in this thesis. Rather than just presenting the porting data, we propose a novel way of representing the cross-project porting activities using four different visualizations.

2.3 Tool Support and Mining Software Repositories for Cross-Project Activities

Mining software repositories can be one of the first steps for identifying clones and relationships between projects as well as for understanding porting activities. Wagstrom et al. developed GitMiner [25] to identify the relationship between projects and to track user activities across projects based on the data mined from the GitHub repository (specifically for Ruby on Rails projects). GitMiner [25] identifies the relationship between Ruby on Rails projects with the other projects in the ecosystem and also facilitates understanding of this mined data in the context of relationships between projects, users, and their activities. Ohira et al. focused on enhancing knowledge collaboration among peer project developers by mining project data in sourceForge using GRAPH-MANIA [17]. This tool promotes knowledge collaboration by allowing its users to access the cross project and developer information.

Although many of the previous studies referenced in this section attempted to resolve issues related to cross-project activities, none of the existing awareness tools aimed to represent porting across projects similar to how this thesis attempts to visualize the activities of BSD project family. This work focuses on the development of a visualization tool that utilizes patch pairs generated by REPERTOIRE [18], attempting to display cross-project porting activities using a web interface. The main goal of this thesis is to help novice developers and contributors understand cross-project porting activities.

2.4 Visualization Aid for Cross-Project Activities

Software visualization and metrics are two powerful techniques for representing historical data. Visualizing enormous amounts of information is particularly useful for easily exploring data. Canfora et al. proposed a novel method, Cross-System-Bug-Fixing (CSBF) [4], for investigating the propagation of related changes and bug fixes across cross projects. They have used the clustered social networks of OpenBSD and FreeBSD committers following the CSBF method. They showed that less than *10%* of the OpenBSD and FreeBSD committers were the same; only *10%* of the changes traceable to source projects and contributors appeared in email discussions. Biehl et al. developed FASTDash [3] in order to visualize the following: current development activities, the methods and files being modified, and the active developers in these activities. These projects primarily aimed to increase the awareness of current activities rather than to analyze repository data. Their user study and interviews showed that visualization improves team awareness; however, one limitation of FASTDash is that it cannot preserve visually analyzed data for future use. Ueday et al. developed Gemini [24] to visualize the code clones detected by CC-finder, while Lanza et al. used code crawler [15] to present lightweight polymetric views of the source code.

Although the works referred to above used visualization to represent various activities and results, most were limited to visualizing either socio-technical relations or clone detection results, or to tracking bug-fix activities. In contrast, the current work displays historical evolution data to improve

awareness of past porting activities. It also preserves the results and provides a way of accessing the results via web browser. The tool uses four different visualizations to provide a birds-eye view of porting activities based on four different associations (file level, patch-file etc.).

2.5 Web-based Tools for Analyzing Software Evolution

Web-based tools are easily accessible, require no configuration set-up, and are ideal for the globally-distributed open-source developer community. Developed by D'Ambros et al., Churrasco [7] visualizes and analyzes software evolution through a web interface framework. These researchers studied the impacts of migrating the desktop software to a web platform using the SPO (Small Project Observatory) [6] and the Churrasco framework [7]. They analyzed the availability, scalability, and browser compatibility of this newly-migrated web tool. Surveys conducted on their tool showed that *50%* of the users agreed that the web framework was useful, and *72%* of the users agreed that collaboration is important in reverse engineering tasks. Similar to the above works, this thesis attempts to improve awareness of cross-project porting activities among OSS developers. We have developed VIGNETTE, a web-based tool to visually analyze various aspects of cross-project porting.

Chapter 3

Tree Visualization

This chapter introduces the features of *tree* visualization with Section 3.1 presenting the background for the tree view. Section 3.2 provides details about the employed color scheme, while Section 3.3 shows input data formats. Section 3.4 describes the interactive features of the visualization, and Section 3.5 discusses the potential benefits of the tree view.

3.1 Background

It remains a time-consuming and tedious task to identify the source of cross-project porting activities using the raw project data in the repository. We thus aimed to easily identify the sources of porting activity and to track porting patches. For example, suppose a project file introduces a new defect after porting a feature enhancement. Knowing the sources of the ported patch could help the developers apply the bug fix patch consistently and easily track the updates to the patch. For example, suppose a bug fix patch was applied to the project file *ap/A.c* of FreeBSD and was ported to the project file *ic/A.c* (similar name) of NetBSD. After this porting activity, if the FreeBSD developer identifies that the defect was not completely fixed and updates the bug fix

patch. In this situation, the NetBSD developer who has ported the patch from the FreeBSD file can use this visualizations to track for the patch updates (as he is aware of the porting activity of the FreeBSD file with a similar file name). The updates to the patch ids can be tracked using the latest patch commit date presented in the Additional information page of this visualization along with the hierarchical representation.

3.1.1 Motivation

Suppose a novice developer has a bug fix task to complete. If cross-project porting activities are prevalent in the project, the developer may need to know the modules from the peer project that could possibly be affected by or may affect the bug fix. Searching through the project repositories could help the developer to locate the affected peer projects, but this search represents a daunting task. Moreover, the developer could not identify all affected projects with a manual search through the repositories. Tree visualization provides an alternative way to quickly identify all affected projects and could also help the developer focus on the bug fix task itself.

3.1.2 A Hierarchical Representation

Tree visualization represents the projects in a hierarchical form. The depth of the hierarchical tree reflects the actual project folder's structure in FreeBSD and NetBSD projects. For example, if a file *foo.c* has a path *sys/ids/foo.c* from the project root folder (freeBSD), then VIGNETTE dis-

plays *foo.c* in a tree with *sys* as a root node. The current implementation of the visualization shows the file level association for FreeBSD and NetBSD project files. Figure 3.1 shows a sample hierarchical view of FreeBSD sub-projects.

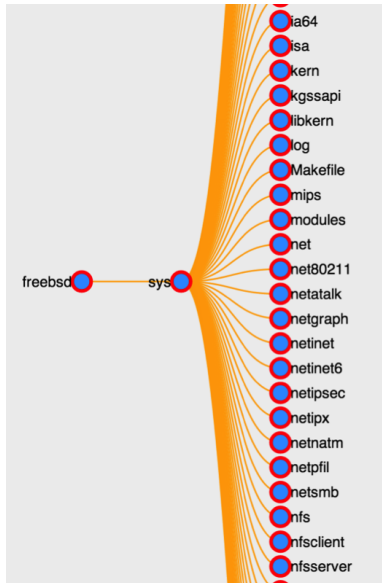


Figure 3.1: FreeBSD hierarchical tree view

Node. A *node* represents a file, module, directory, or subdirectory of a project. Every node displays the corresponding text (file name) near the node circle. Color filling can be used to differentiate between nodes having child nodes and nodes without child nodes. A parent node represents a top-level folder or directory in a project structure.

3.1.3 Comparison View

To compare cross-project porting activities between two projects for the selected year, this page is built with two different frames. Each frame displays the hierarchal tree view for a project. The current implementation of the page represents the first frame with FreeBSD data and the second frame with NetBSD data. The affected files of the NetBSD projects gets displayed automatically in the second project frame when the user selects a node (in the first frame) by clicking on it. Figure 3.2 displays a sample snapshot of the tree visualization page showing the comparison view.

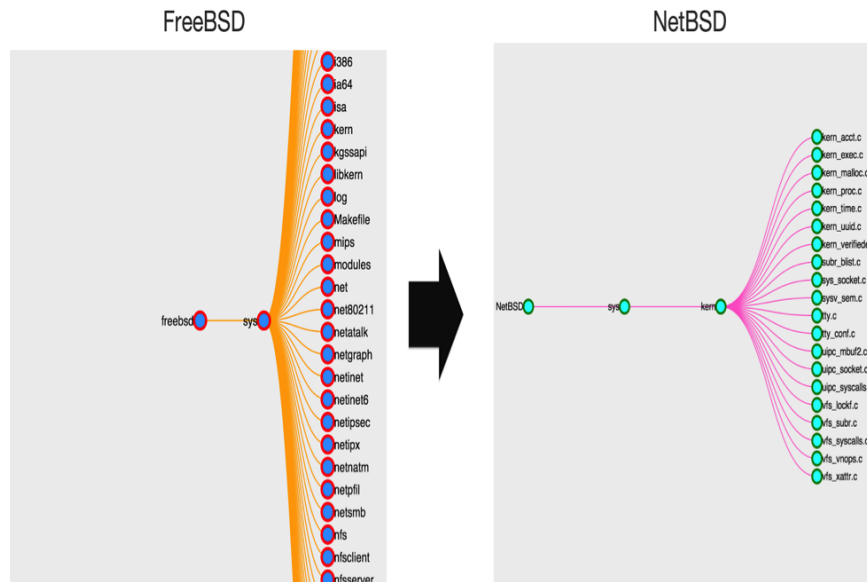


Figure 3.2: Comparison views of FreeBSD and NetBSD projects

Year Selection Drop Down. The year selection drop-down allows the user to select the year of interest for porting activities. Values for the year

selection drop-down range from 1997 to 2014.

BSD Project Selector Drop Downs. The two BSD project drop downs can be used to select the two different projects for comparing the cross-project activities. Both the BSD drop-downs have three options (FreeBSD, NetBSD, and OpenBSD).

3.2 Coloring Scheme

We used different colors to represent different project elements in the visualization. Table 3.1 describes the color scheme and its interpretation.







S. No.	Color	Interpretation
1.		Represents a FreeBSD project node
2.		Represents a FreeBSD edge
3.		Represents a compressed FreeBSD node with children
4.		Represents a NetBSD project node
5.		Represents a NetBSD edge
6.		Represents a compressed NetBSD node with children

Table 3.1: Tree visualization coloring scheme interpretation

3.3 Data Formatting

The format of the data used to draw the graph is very important when using Data-Driven Document JavaScript (D3.js) API. Data can either be hard-coded or loaded from external resources. Binding data from an external resource offers the advantage of handling data dynamically. The D3 API parses and binds data in any one of the following formats:

- XMLHttpRequest
- Text file
- JSON blob
- HTML document fragment
- XML document fragment
- Comma-Separated Values (CSV) file
- Tab-separated values (TSV) file

We developed the current graph by binding data from the database in a JSON blob format. VIGNETTE queries the database and retrieves all the patch ID's for the selected FreeBSD file (in the FreeBSD frame) for the specific year. Next, it queries for all the affected NetBSD file paths using the retrieved FreeBSD patch IDs. Then, it converts these file paths to a hierarchical JSON format and writes the formatted data to a file, which is later used to display

the graph in the NetBSD frame. The view presented in the NetBSD frame represents the cross-project porting activity based on the file name similarity.

3.3.1 Sample Data with Example

If a user selects a node with the name *kern* in the FreeBSD frame after selecting the year as 2003, then a query will be sent to the database as soon as the user selects the cancel button on the pop-up dialogue box. Then VIGNETTE retrieves the affected NetBSD projects files for *kern* in 2003 using the aforementioned steps. The following are the sample NetBSD folder paths retrieved from the database, which are formatted to the JSON format to generate the tree view for NetBSD (refer to figure 3.2).

List of folder paths:

NetBSD/sys,
sys/kern/uipc_socket.c,
sys/kern/vfs_syscalls.c, sys/kern/uipc_syscalls.c,
sys/kern/sys_socket.c, , sys/kern/vfs_syscalls.c,
sys/kern/vfs_xattr.c, sys/kern/vfs_syscalls.c,

3.4 Interactive Features

This section discusses various interactive features supported in this visualization.

3.4.1 Graph Zoom In and Zoom Out

The tree view API has a built-in zoom in and zoom out feature. To zoom in and zoom out on the graph, the mouse focus should first be brought inside the frame; then, the mouse can be scrolled in and out respectively.

3.4.2 Graph Panning

The user can change the view position via a mouse operation, another flexible feature of this visualization.

3.4.3 Click Event

Clicking any node in the graph displays a pop-up dialogue box and performs the following operations: (1) Submit the tree visualization form to initiate a query for cross-project data if the user selects the *cancel* button, or (2) Redirect the user to an additional information page if the user selects the *ok* button. Figure 3.2 shows the resulting affected NetBSD project files for the cancel button selection, while Figure 3.3 shows the pop-up dialogue box.

3.4.4 Additional Information Page

We designed the Additional Information page with the aim of giving the developers an opportunity to further investigate the porting activity associated with the selected module. Using the Additional Information page, novice developers can learn more about the developers involved in the activity, their email IDs, the list of used patch IDs and the activities' dates. Figure 3.4

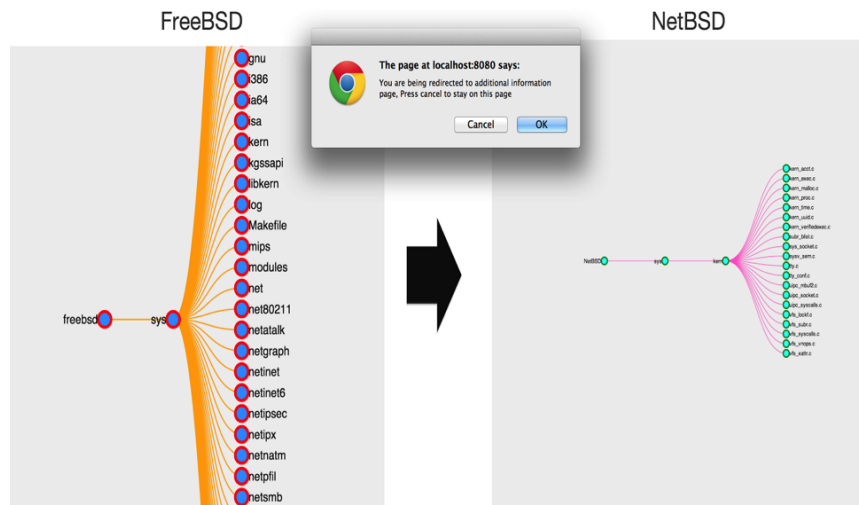


Figure 3.3: Pop-up dialogue box displayed when a node is clicked

provides an example snapshot that captures all of the additional information pages.

3.4.5 Learn More Page

Clicking on the “Learn more” button (shown in Figure 3.5) displays a tutorial on tree visualization (shown in Figure 3.6).

3.5 Intended Benefits

The benefits of using the tree visualization include the following: (1) developers can easily learn about corresponding affected files in a peer project by clicking a node in the tree visualization, and therefore (2) they can be more productive by focusing on development tasks.

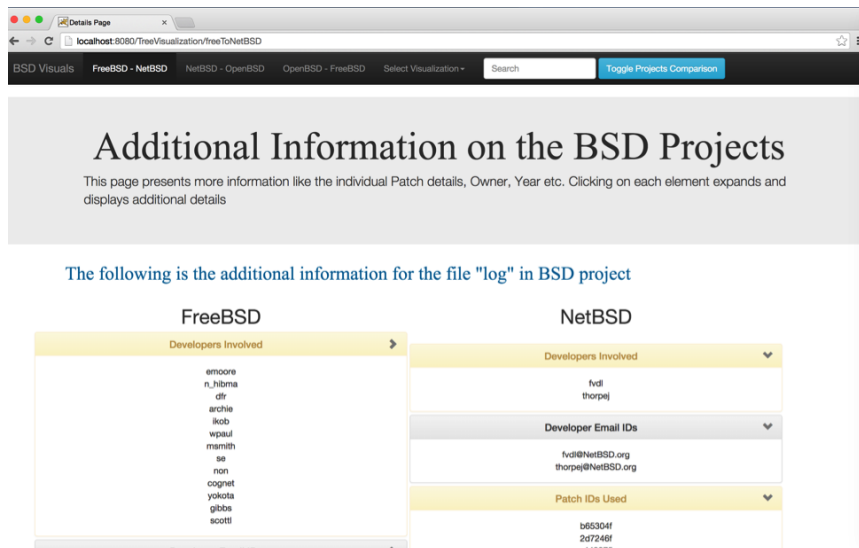


Figure 3.4: Tree visualization additional information page

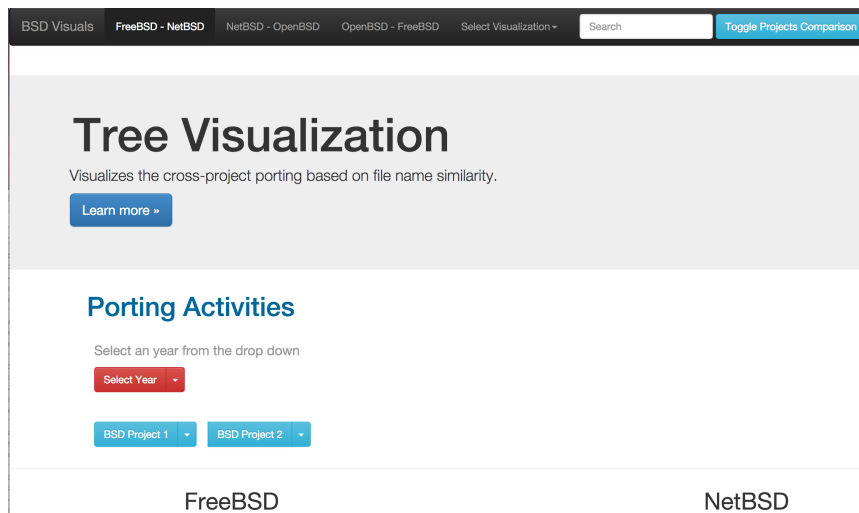


Figure 3.5: Learn more button

Learn More About the Tree Visualization!

About Tree Visualization

Introduction

Tree visualization shows a file level association of cross-project porting activities. The file level association can be defined as follows. For instance, applying a patch, say P1, modifies a FreeBSD file. If P1 was adapted to a file with a similar name in NetBSD, then the two files are associated. Knowing such cross project porting activities is important for the user(developer) in situations where a patch update or a bug fix is to be made to the same set of files. Instead of searching through the entire repository to know the files affected by patch Id's, the users can use this visualization to find the affected files based on the year of porting. Moreover, all the user needs to perform is a simple click operation on the nodes of the tree graph to find out the affected files in the peer project. Additionally, the developer can further investigate by navigating to the additional information page to know all the relevant details about the patch Id's that were applied to the peer project files.

How to use Tree visualization?

Using Tree visualization is very simple. The following are the steps involved to get a file level associations across the peer projects.

Step 1: Navigate to the Tree visualization from the landing page by clicking view button in the Tree visualization thumbnail

Figure 3.6: Tree visualization learn more page

Chapter 4

Bubble Chart Visualization

This chapter introduces the features of *bubble* visualization. Section 4.1 describes the background for implementing the bubble visualization, while Section 4.2 provides an overview of the chart. Section 4.3 interprets the bubble chart color schemes, and Section 4.4 discusses the bubble chart's input data format. Section 4.5 describes the interactive features, while Section 4.6 discusses the intended benefits.

4.1 Background

Porting is an important activity in software evolution, yet few porting awareness tools exist. In an attempt to emphasize the importance of porting activities, the bubble chart visualization measures and displays porting activity frequency over the entire history of BSD projects.

4.1.1 Motivation

Suppose a novice developer has joined the BSD community and is totally unaware of the porting activities in the BSD projects. In an open-source development environment, these activities cannot be learned from peer devel-

opers due to the geographically-distributed nature of the community and also due to the split in the developer community. Bubble visualization represents an alternative way to quickly learn about cross-project porting activities.

4.2 A Bubbled Representation

Bubble visualization represents the porting activity frequency for three BSD projects taken pairwise. The current implementation of this visualization shows the frequency of porting only for three BSD projects (FreeBSD, NetBSD and OpenBSD) for the entire project history (1993 to 2013). Figure 4.1 shows the frequency of porting for FreeBSD-NetBSD, OpenBSD-FreeBSD, and NetBSD-OpenBSD.

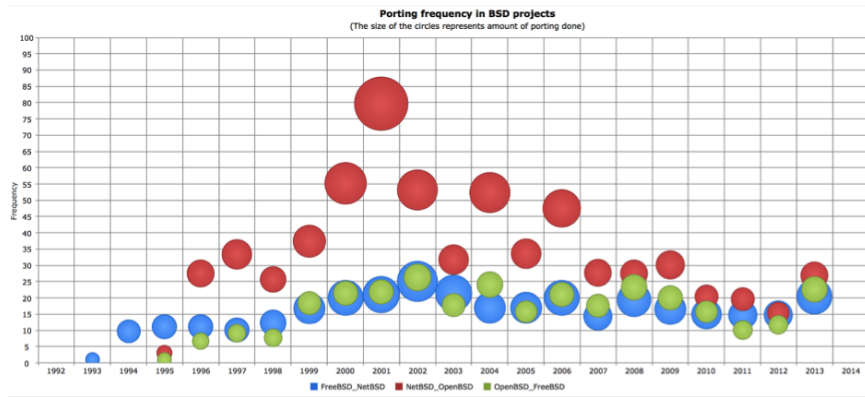


Figure 4.1: Porting frequency in bubble visualization

Bubble. The chart shows one bubble per year for each project pair displaying the porting frequency. We define porting frequency as the number of times the patch in one project is applied to a different project. For example,

if patch P was released and applied to project A, and project B has ported the patch from A, then we consider project A as the source of the porting activity.

X-Axis. The x-axis represents the years of porting activity. The unit interval on the x-axis is one year. Currently the chart records frequencies from 1993 to 2013.

Y-Axis. The y-axis represents the porting activity count. The unit interval on the y-axis is 2.

Bubble Weight. The size or weight of the bubble varies based on the porting frequency. A large amount of porting frequency results in a larger bubble. The following represents the porting frequency scale on a graph. If the actual frequency of porting is 100 then it is represented as 1 on the graph.

4.3 Coloring Scheme

We represent the porting frequencies of the three project pairs (FreeBSD-NetBSD, NetBSD-OpenBSD, and OpenBSD-FreeBSD) with unique colors. Table 4.1 provides the color scheme interpretation.

4.4 Data Formatting

We provide the input data format of the bubble chart as an array of JSON. As attributes for the three projects, the data includes the year, porting activities count and weight parameters.




S. No.	Color	Interpretation
1.		Porting source as NetBSD
2.		Porting source as OpenBSD
3.		Porting source as FreeBSD

Table 4.1: Bubble visualization coloring scheme interpretation

4.4.1 Sample Data with Example

VIGNETTE generates the bubble chart by parsing the key value pairs. The data has the following as keys: *year*, *FreeBSD*, *NetBSD*, *OpenBSD*, *WFreeBSD*, *WNetBSD* and *WOpenBSD*. From the data input, the tool parses the year variable values and porting activity count (FreeBSD, NetBSD, and OpenBSD) values and represents them as (x , y) coordinates respectively. Then it assigns the bubble size based on the value parsed from *WFreeBSD*, *WNetBSD* and *WOpenBSD* porting frequency variable.

4.5 Interactive Features

This section discusses various interactive features supported with this visualization.

4.5.1 Bubble Mouseover

We built the bubble chart with a mouse-over feature. When the user hovers over the bubbles, a tool-tip with additional information is displayed. The tool-tip displays the porting source, the porting year, and frequency, as illustrated by figure 4.2.

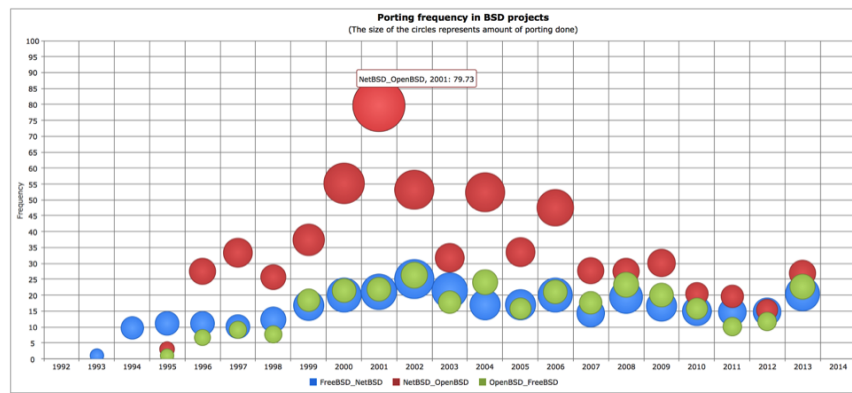


Figure 4.2: Bubble mouse over event

4.5.2 On Bubble Click

Bubbles in the chart are clickable. Clicking on any of the bubbles will display a redirection pop-up window. Selecting the *cancel* button on the pop-up dialogue box displays the bubble chart page. Selecting the *ok* button on the pop-up dialogue box redirects the user to the Additional Information page. Figure 4.3 is a sample screenshot showing the pop-up dialogue.

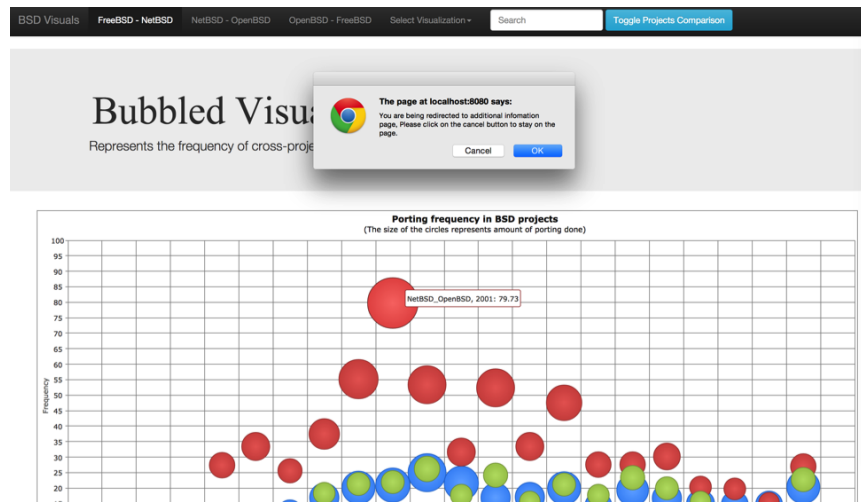


Figure 4.3: Bubble mouse click event

4.5.3 Additional Information Page

The Additional Information page enables the developer to further investigate the files that have contributed to the frequency of porting for the chosen year. The Additional Information page displays the Patch ID applied, developers involved, and files for the selected project pair. Figure 4.4 displays the Additional Information page for the bubble visualization.

4.5.4 Learn More Page

Clicking on the “Learn more” button (shown in Figure 4.5) displays a tutorial on bubble visualization (shown in Figure 4.6).

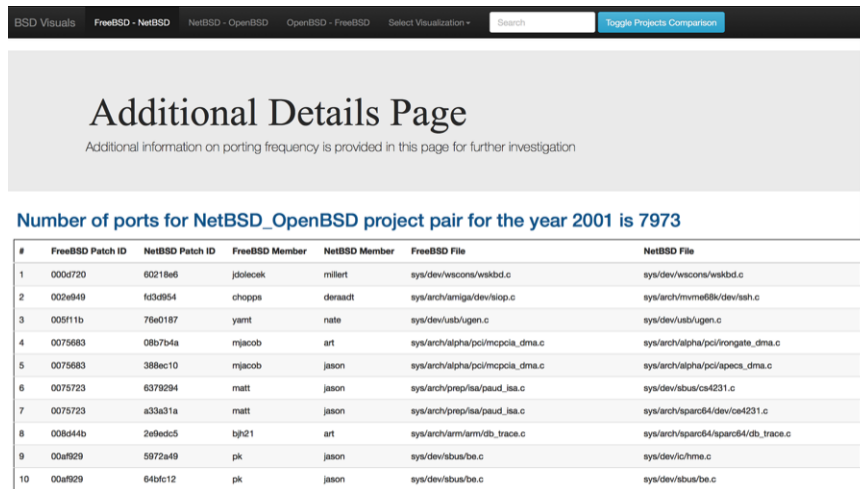


Figure 4.4: Bubble visualization additional information page

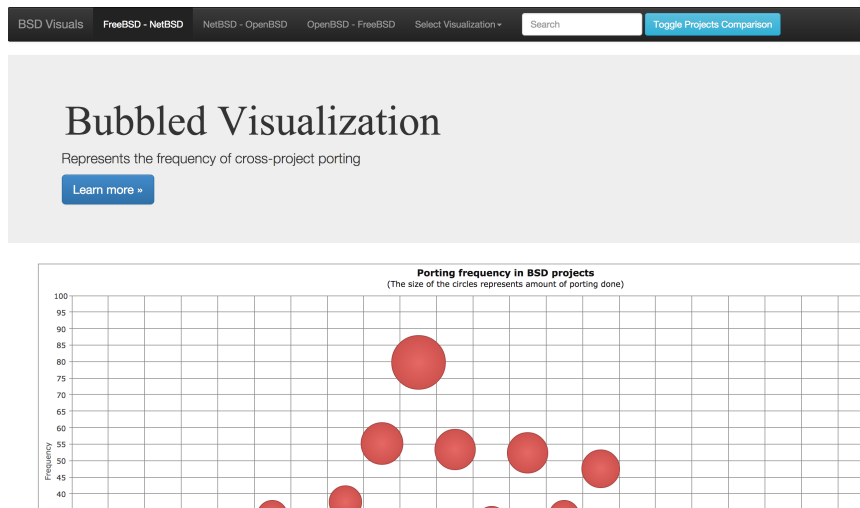


Figure 4.5: Learn more button

4.6 Intended Benefits

The benefits of using the bubble chart visualization include the following: (1) novice developers can quickly learn the importance of porting



Figure 4.6: Bubble visualization learn more page

activities, (2) developers can leverage the data from the Additional Information page to support their maintenance tasks, and (3) managers can use this view to improve the open-source project process.

Chapter 5

Developer Dependency Wheel Visualization

This chapter introduces the features of the *developer dependency* visualization. Section 5.1 presents a discussion of the background, and Section 5.2 describes the purpose of different web page elements. Section 5.3 interprets the color scheme used in the chart, and Section 5.4 provides more detail on the data format used to build the chart. Section 5.5 describes the interactive features, and Section 5.6 outlines the intended benefits.

5.1 Background

Open source developers mostly rely on mailing lists and forums to learn all the developers whose modules may be affected in a porting activity. Obtaining this list is difficult (due to forum access restrictions) if cross-project developers are involved in the porting activity. To address this difficulty, we developed a developer dependency wheel in this thesis. This visualization analyzes and displays the cross-project developer associations based on past porting activities.

5.1.1 Motivation

Suppose developers ported patch P across different projects and later updated this patch. If this patch update is very important and has to be communicated to different cross-project developers. One way of getting the list of developers is by using the discussion forums but the forum access restrictions may not allow the cross-project developer to know the developers involved. In this situation, the developer can get the list of the cross-project developers involved in a porting activity along with him/her by using the connecting chords in the dependency wheel visualization. The connecting chords show that the developers were involved in the same porting activities for the selected year. The current implementation shows the associations for the FreeBSD and NetBSD developers.

5.2 Developer Dependency Wheel

In this visualization, we represent the chart as a circular wheel with connecting chords. We built the circular wheel with a number of arcs, each one representing a developer from either FreeBSD or NetBSD project. We associated developers in the chart based on their common porting activity participation.

Arc Wheel. The arc length varies with the number of times a developer was associated with a peer project developer. Figure 5.1 shows the variable length arcs for each developer. For easy identification, we prefixed the FreeBSD developers with “*FBsd/*” to their names and the NetBSD developers

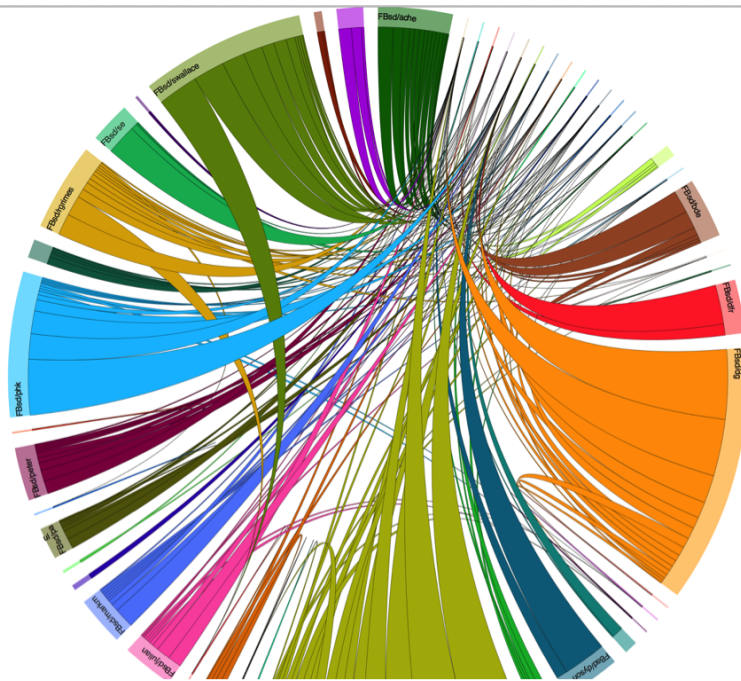


Figure 5.1: Developer dependency visualization

with “*NBsd/*” to their names.

Connecting Chords. Chords connecting the two arcs of the wheel show the developers’ associations. We determined the width of a connecting chord by the number of times the same developer was paired in different porting activities. Figure 5.1 displays the pairing of the developers connected by chords.

The dependency visualization has a common navigation bar, title, description ribbon and year selection drop-down. It also has a back button at the bottom of the page. For simplicity and scalability purposes, the current implementation shows the association between 50 peer project developers for

the year 1995. Figures 5.2 and 5.3 represent sample screenshots that capture all page elements.

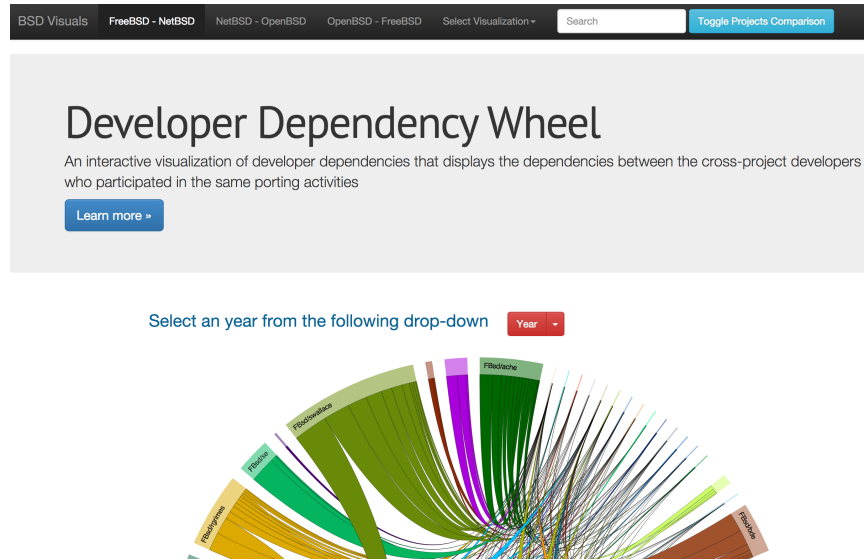


Figure 5.2: Developer dependency wheel web page

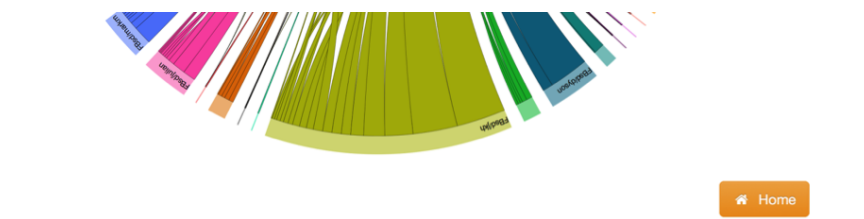


Figure 5.3: Developer dependency page showing back button

5.2.1 Learn More Page

Clicking on the “Learn more” button (shown in Figure 5.2) displays a tutorial on developer dependency wheel visualization (shown in Figure 5.4).



Figure 5.4: Learn more page

5.3 Color Scheme

Each developer can be identified through a unique color in the chart. In the current chart implementation, we used 50 unique color codes to represent 50 developers in both FreeBSD and NetBSD projects.

5.4 Data Formatting

This chart requires the input data to be formatted as a dependency matrix. We modeled data obtained from the database as a dependency matrix. Rows and columns in the matrix represent FreeBSD and NetBSD developers. Elements in the matrix have a value of zero, if the developer pair has not participated in the same porting activity; a value of greater than zero represents

the number of times of pairing. Suppose developer A of FreeBSD is mapped to a column index 1, and developer B of NetBSD is mapped to a row index 2 in the dependency matrix. If the number of times A and B were paired is 10, then the element identified as dependency [2][1] will be updated to 10. Figure 5.5 illustrates a sample dependency matrix for the 50 developers of FreeBSD and NetBSD projects.

```
[
[2,0,0,1,0,0,1,0,1,1,0,1,0,1,1,1,0,1,0,0,0,0,1,0],
[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1],
[0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,1,0,1],
[0,0,0,0,0,0,6,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,1,1],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,5,0,1],
[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,7,1],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,2],
[1,0,0,1,0,0,4,0,0,1,0,1,0,1,1,1,0,1,0,0,0,5,2,1],
[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,7,4,1],
[0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0],
[0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,1,3,0,2],
[0,0,2,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,3,1,2,0],
[1,0,0,1,0,0,1,0,1,1,0,9,0,0,1,1,0,1,1,0,0,0,0,1],
[1,0,0,1,0,0,1,0,1,1,0,1,0,1,0,1,0,1,1,0,0,3,1,0],
[1,0,0,1,0,0,1,0,1,1,0,1,0,7,1,0,0,1,0,0,0,0,2,1],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1],
[0,0,1,1,0,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,2,0],
[0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,2,1,0],
[0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,2,0,1],
[0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,9,1,0,1]
]
```

Figure 5.5: Dependency matrix for 50 FreeBSD and NetBSD developers

5.5 Interactive Features

This section provides the interactive features of the developer dependency wheel visualization.

5.5.1 Mouseover

Wheel Arc. Hovering on the arc displays a tool-tip with additional information, including the developer’s name and the percentage of the developer’s porting participation. Figure 5.6 shows a tool-tip displaying that a FreeBSD developer “Ache” was actively paired for *56.0%* of the time with different NetBSD developers.

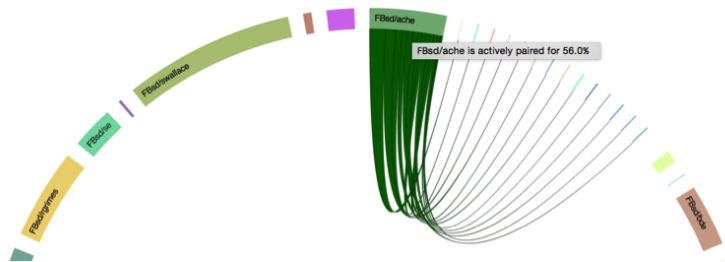


Figure 5.6: Mouse over action on wheel arc

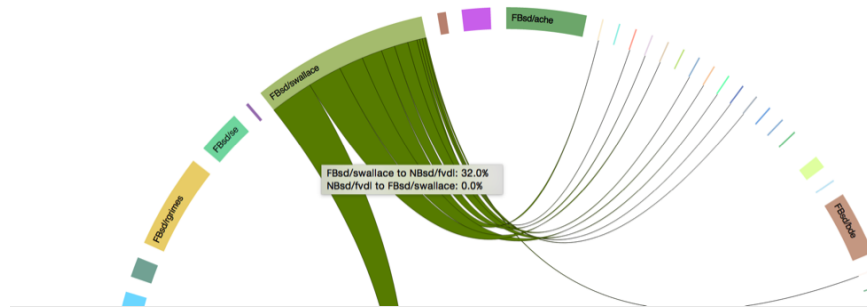


Figure 5.7: Mouse over action on wheel chord

Connecting Chord. Hovering over the chord displays a tool-tip with additional information, the percentage of developer pairing. Figure 5.7

shows a tool-tip displaying how FreeBSD developer “*Swallace*” was paired with NetBSD developer “*Fvdl*” in the same porting activity for 32% of the time.

5.6 Intended Benefits

The benefits of using the dependency visualization include the following: (1) developers can gain a great deal of developer association information from the compact representation, and (2) they can reduce the time spent on obtaining the list of cross-project developers.

Chapter 6

Bipartite Visualization

This chapter introduces the features of our *bipartite* visualization. Section 6.1 introduces the background motivation, and Section 6.2 describes the bipartite visualization. Section 6.3 interprets the color schemes used. Section 6.4 discusses the data format. Section 6.5 describes the interactive features of the graph, and Section 6.6 presents the intended benefits.

6.1 Background

As software evolves, developers may need to apply an updated patch to the same project and peer projects files. Knowing all of the dependent project files can be tedious and error prone. Bipartite visualization overcomes this issue by identifying the files that have the same patch applied across cross-projects.

6.1.1 Motivation

Suppose different developers apply the patch with id *f123* to two different projects of the FreeBSD project and one project of the NetBSD project. When patch P is updated, the updated patch may be applied to all the af-

affected project files. It is easy to identify the affected files, if the patch was applied to a small number of files. However, if the patch was applied to many project files, the developers may overlook updating some of the affected files. To address this issue, we have developed the bipartite visualization. Using this visualizations developers will know all the files (even with different names) that have used the same patch.

6.2 Bipartite View

Bipartite visualization displays the patch-file relationship for a given patch applied to the files across peer projects. This view has two pairs of stacked bars showing these patch-file relationships. In each pair of the stacked bar, the left-hand sidebars represent the patch IDs, and the right-hand sidebars represent the file names. The current implementation of Bipartite visualization shows the relationship of FreeBSD patch Ids with that FreeBSD files and also the NetBSD files.

6.2.1 Comparison View

The comparison view gives the developer an opportunity to view and compare two different patch-file relationships. The bipartite chart provides two patch-file relationships, showing the FreeBSD patch ID to FreeBSD file as well as the FreeBSD patch ID to the NetBSD file. Figure 6.1 shows a comparison view with two patch-file relationships. They represent either the patch IDs or file names across the same or peer project.

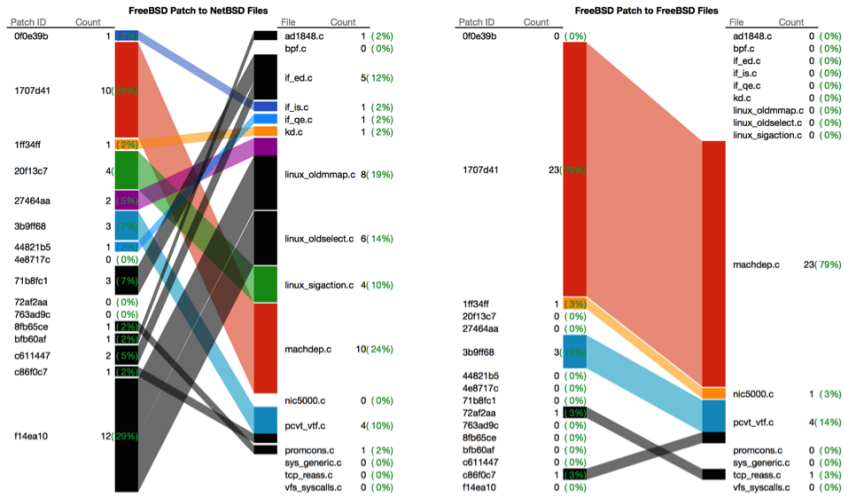


Figure 6.1: Bipartite comparison view

Stacked Bars. They represent either the patch IDs or file names across the same peer project.

Connector. The translucent connectors map the patch IDs to the file names.

Percentage. The number of times the same patch ID was used is expressed as a percentage on the visualization.

Count. The count on either side of the graph represents the number of usages for a particular file or patch.

Page Elements. The bipartite view has a common navigation bar, a title, a description ribbon, a learn more button, a year drop-down and a back button. The current implementation visualizes 20 patch-file relationships

which were randomly selected for the year 1995. Figure 6.1 shows a sample screen for the bipartite visualization page.

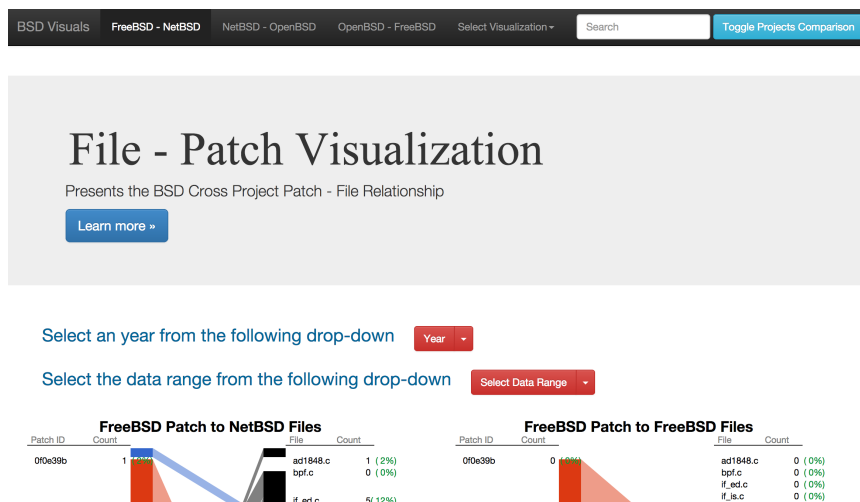


Figure 6.2: Bipartite visualization web page

6.2.2 Learn More Page

Clicking on the “Learn more” button (shown in Figure 6.2) displays a tutorial on bipartite visualization (shown in Figure 6.3).

We randomly assigned a color to each bar representing a patch ID or file name from a defined set of color codes.

6.3 Data Formatting

We generated the bipartite view by parsing the data represented as an array of tuples. We constructed each array with four attributes: the *patch ID*,

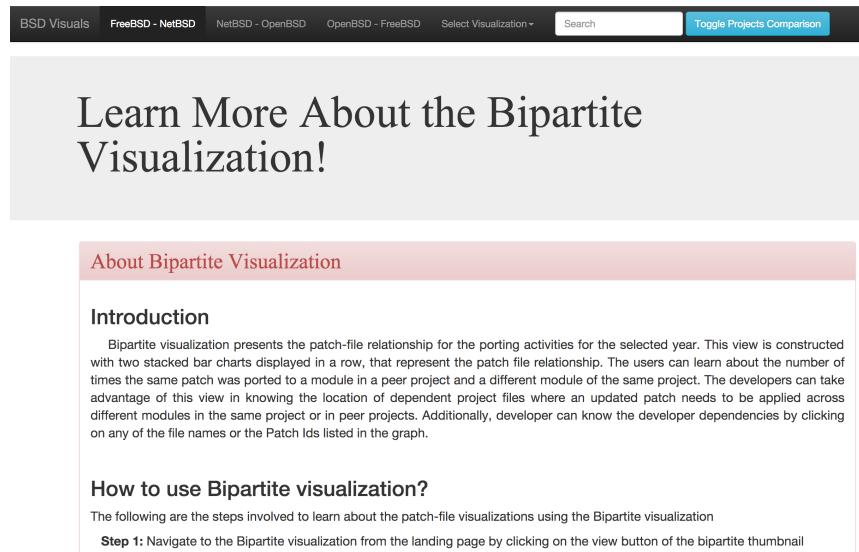


Figure 6.3: Learn more page

patch count, *file name*, and *file count*. We also constructed each tuple with four attributes. Figure 6.4 shows data formatted as an array of tuples.

```
[ 'f14ea10', 'linux_oldselect.c', 6, 0 ],
[ '1ff34ff', 'kd.c', 1, 0 ],
[ '1707d41', 'machdep.c', 10, 23 ],
[ '8fb65ce', 'sys_generic.c', 0, 0 ],
[ 'f14ea10', 'linux_oldmmap.c', 6, 0 ],
[ '72af2aa', 'tcp_reass.c', 0, 1 ],
[ 'c86f0c7', 'pcvt_vtf.c', 1, 1 ],
[ '4e8717c', 'vfs_syscalls.c', 0, 0 ],
[ '763ad9c', 'bpf.c', 0, 0 ],
[ '44821b5', 'if_qe.c', 1, 0 ],
[ '3b9ff68', 'pcvt_vtf.c', 3, 3 ],
[ '27464aa', 'linux_oldmmap.c', 2, 0 ],
[ '71b8fc1', 'if_ed.c', 3, 0 ],
[ 'bfb60af', 'ad1848.c', 1, 0 ],
[ 'c611447', 'if_ed.c', 2, 0 ],
```

Figure 6.4: Bipartite visualization input data format

6.4 Interactive Features

This section discusses various interactive features of the bipartite view.

6.4.1 Mouse Over

It is possible to hover over the stacked bars in this visualization. When hovering over a stacked bar, thus a user highlights the patch file relationships in both parts of the view, and the remaining connectors disappear. Figure 6.5 displays the mouse-over operation.

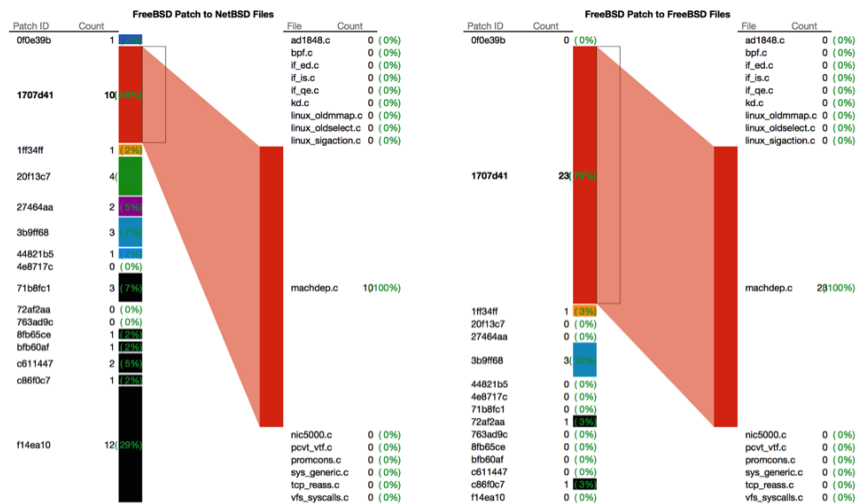


Figure 6.5: Bipartite chart mouseover on the patch ID

6.4.2 Mouse Click

Clicking the patch IDs displays a pop-up dialogue box. Selecting the *cancel* button of the pop-up dialogue box maintains the bipartite view. Choos-

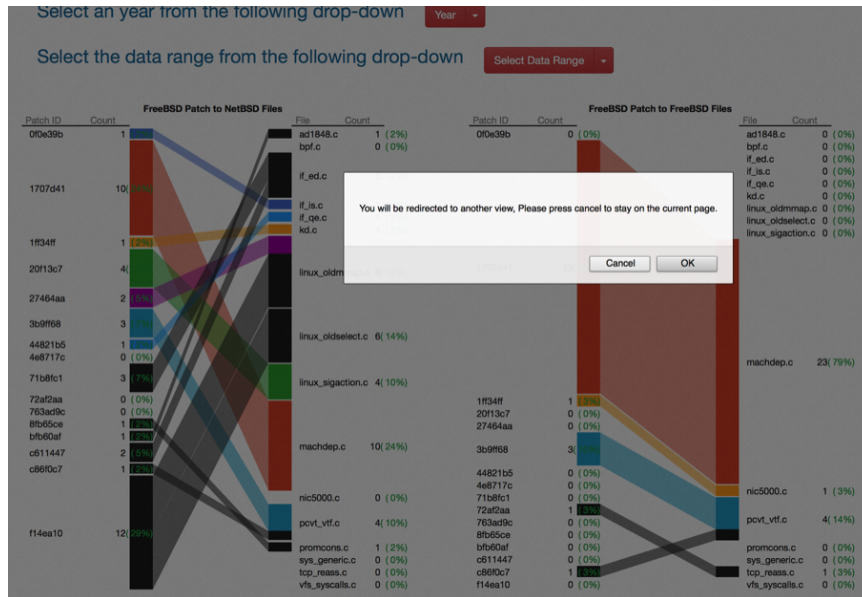


Figure 6.6: Bipartite view click operations

ing the *ok* button of the pop-up redirects the user to the developer dependency wheel. Figure 6.6 shows a pop-dialogue box for a click event.

6.5 Intended Benefits

Knowing the patch-file pair of a past porting activity can help the developer to consistently apply an updated patch.

Chapter 7

Implementation

This chapter gives the implementation details for VIGNETTE. Section 7.1 discusses various web pages. Section 7.2 discusses the APIs used to build the application.

7.1 Web Pages

The web application includes the following web pages: the Welcome page, the Learn More page, and four visualization pages.

Welcome page. The welcome is a landing page for the application. As soon as the user accesses the application URL, he or she sees the home page displayed. Figure 7.1 displays the common navigation bar. The *Select Visualization* drop-down on the navigation bar allows users to navigate between different visualizations. Each page displays a title ribbon. From the home page, the user can navigate to each of the visualizations using the thumbnails, as displayed in Figure 7.2.

Learn more page. Users can access this page by clicking on the *Learn More* button in the title ribbon of the home page. This page gives a brief motivation for the development of VIGNETTE. A back button can be

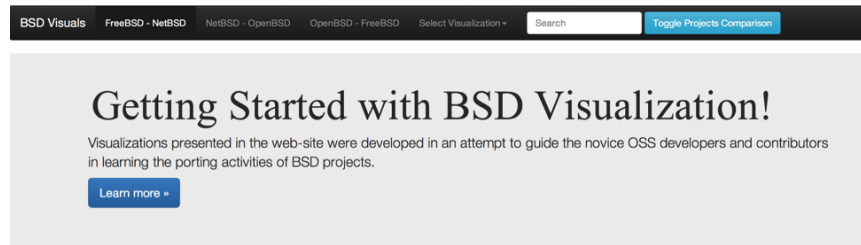


Figure 7.1: Common navigation and page ribbon

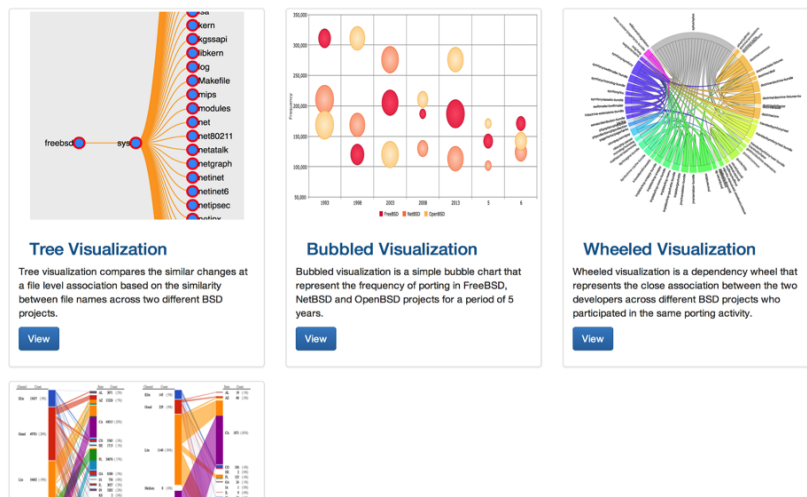


Figure 7.2: Thumbnail views of visualizations

used to navigate back to the Welcome page. Figure 7.3 shows the Learn More page.

7.2 APIs and Server

We built VIGNETTE using many external APIs and developed the application's back-end with servlets and JSP (Java Server Pages) technology.

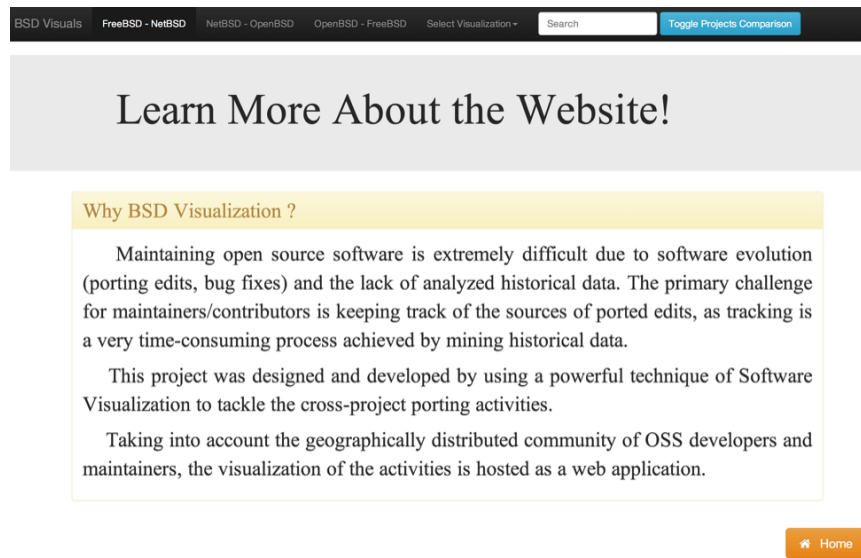


Figure 7.3: Learn more page

We stored the porting data in a MySQL database. We developed the front-end for the application using JavaScript, JQuery, HTML and CSS. We built three of the visualizations (tree view, developer dependency wheel, bipartite visualization) using the Data-Driven Document (D3.js) JavaScript API. We deployed the application in a remote Tomcat web server. The application can be accessed using the URL below:

<http://www.vignettetool.com/TreeVisualization>

Data-Driven Document (D3) library. D3 is a JavaScript library used to manipulate DOM elements with a data-driven approach. Data is bound to the DOM element using HTML, CSS3, and SVG (Scalar Vector Graphics). An important feature of D3 is its ability to render the visualization

as SVG elements. An SVG element is preferred to an HTML `<div>` element, as rendering the data with `<div>` is not consistent across different browsers. Therefore, D3 API is reliable, visually consistent, and faster. D3 introduces dynamism to the visualizations and supports animations.

Browser Compatibility. Users can access the web application via any of the browsers. We have tested VIGNETTE in Google Chrome, Safari and Firefox browsers.

Chapter 8

Evaluation and Results

This chapter describes the methodology used to evaluate VIGNETTE and presents the evaluation results. We designed and conducted a user study to evaluate VIGNETTE with the aim of analyzing its usefulness.

8.1 User Study Design

We conducted an initial user study with two participants. Before performing any tasks, we introduced the users to the concept of cross-project porting, including an exploratory demo of VIGNETTE. We also provided the participants with a user study reference manual (presented in Appendix A). The user study manual included an introduction to cross-project porting, a brief overview of the web tool, and an introduction to the various visualizations, tasks, and questionnaires.

8.2 Study Experience

This section presents two user study session experiences. We preserve the anonymity of these users in the following manner: *User A* refers to the first participant, and *User B* refers to the second participant. During the

user study, we captured each participant’s activities by screen recording and transcribed the conversations. We asked each participant to perform the same exploratory tasks on VIGNETTE. As a first step in the study, we gave *User A* and *User B* reference manuals and asked them to read the introduction in order to learn about the background of the research. Next, we verbally explained the research background and the details of cross-project porting to both participants. Finally, we provided a demo on how to use VIGNETTE before asking participants to perform the tasks.

8.2.1 User Profiles

User A. She is a doctoral student in software engineering with over five years of development experience. She is interested in contributing to open-source development and has recently started contributing to an open-source project (Shifu ML). This participant completed all the exploratory tasks in approximately 10 minutes.

User B. He is a post-doctoral researcher in software engineering. With over five years of development experience, he has contributed to multiple open-source projects, such as XIO (a programming language for parallel programming) and Wala (a program analysis framework for Java). This participant completed all exploratory tasks in approximately 15 minutes.

8.3 Results

This section summarizes the findings from the two study sessions. The questions provided for each task were either rating questions with multiple choices, open-ended questions, or concrete questions with a single correct answer. To present quantitative results, we interpreted each user’s answers as a score based on the type of question. The ratings question had a maximum score based on the number of options given; for example, if a question had three options, then the maximum score was three, and the minimum score was one. We did not include the open-ended and background questions in the score interpretations. We interpreted the concrete questions to have a score of either 1 (correct answer selected) or 0 (wrong answer selected). We summarized the score interpretations for the four visualizations in four different tables (Tables 8.1 to 8.4).

8.3.1 Interpretation

Tables 8.1 to 8.4 summarize the resulting interpretations for tree, bubble, bipartite, and developer dependency visualizations respectively. In each of the Tables (8.1 to 8.5), the first column includes the questions related to each task taken directly from the reference manual, A_{answer} and B_{answer} are the answers selected by *User A* and *User B* respectively. A_{score} and B_{score} are the score interpretation of the users. T_{score} is the total score of *User A* and *User B*, and M_{score} is the maximum combined score for any given question. Tables 8.5 and 8.6 provide the overall feedback and user backgrounds respectively.

Question.	A_{answer}	B_{answer}	A_{score}	B_{score}	T_{score}	M_{score}
How would you rate Tree visualization in terms of presenting the file level porting association across freeBSD and netBSD projects?	Excellent	Excellent	5	5	10	10
In a role of novice Open Source developer, would you like to use this tool during project development?	Would think about using the tool	Definitely use the tool	2	3	5	6
How would you rate the information provided in the additional details page for learning about the porting activity?	Useful	Extremely useful	4	5	9	10

Table 8.1: Tree visualization results summary

8.3.2 Result Metrics

To evaluate the research questions, we normalized the scores interpreted in Tables 8.1 to 8.4 with the maximum score (M_{score}). Below, we present the visualization evaluation parameters.

Usefulness. Measures how useful the visualizations are for performing cross-project porting activities.

Question.	A_{answer}	B_{answer}	A_{score}	B_{score}	T_{score}	M_{score}
Could you please select the year that has the largest porting frequency for NetBSD-OpenBSD projects pair?	2001	2001	1	1	2	2
After exploring the bubble chart, how much importance needs to be weighed on the learning porting activities?	Highly important	Highly important	4	4	8	8
On a scale of 1 to 5, how would you think the details provided in the additional details page can be utilized for further investigation?	4	3	4	3	7	10

Table 8.2: Bubble visualization results summary

Effectiveness. Measures whether the visualizations are of any value to cross-project porting activities.

Usability. Measures the ease of use and clarity of the porting information provided in the visualization.

We categorized the user study exploration task questions into three cat-

Question.	A_{answer}	B_{answer}	A_{score}	B_{score}	T_{score}	M_{score}
Select the NetBSD file name that has two different Free Patch Ids applied?	lxoldmap.c	lxoldmap.c	1	1	2	2
Select the FreeBSD file name that has two different Free Patch Ids applied?	pcvtvtf.c	pcvtvtf.c	1	1	2	2
Rate the ease with which the largest patch-file relationship could be identified?	Easy	Very easy	3	4	7	8
Who could benefit most from such type of visualization? (Choose all that apply)	Developer, Committer, Maintainer	Manager, Developer, Committer, Maintainer	3	4	7	8

Table 8.3: Bipartite visualization results summary

egories based on the above evaluation parameters. Figures 8.1 to 8.4 illustrate the final results of the four visualizations and Figure 8.5 shows the overall rating for VIGNETTE.

Question.	A_{answer}	B_{answer}	A_{score}	B_{score}	T_{score}	M_{score}
How would you rate the amount of information presented on the peer project developer dependencies?	Extremely informative	Extremely informative	4	4	8	8
How would you rate the developer dependency wheel for knowing the list of affected developers while a patch is applied to a paired developer's work?	Very helpful	Extremely helpful	3	4	7	8

Table 8.4: Developer dependency visualization results summary

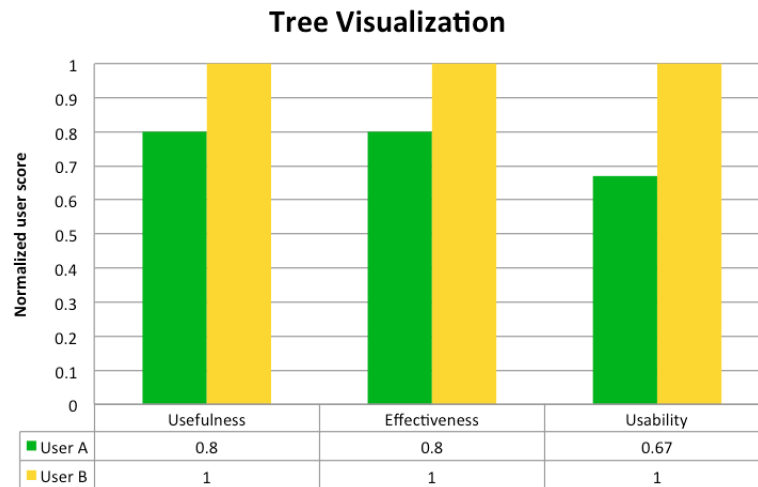


Figure 8.1: Tree visualization results

Question.	A_{answer}	B_{answer}	A_{score}	B_{score}	T_{score}	M_{score}
How would you rate the overall user experience (Look and Feel) of the web tool?	5 (Excellent)	5 (Excellent)	5	5	10	10
Is visualization a powerful medium for learning porting? (Please rate on a scale of 1 - 5 using the following)?	5 (Extremely powerful)	5 (Extremely powerful)	5	5	10	10
Select the best/most useful visualization from the four visualization of the web application?	BiParate Visualization	BiParate Visualization	0	0	0	0
How would you like to rate the ease of use (Navigation between pages) of the web application?	4 (Very easy)	5 (Extremely easy)	4	5	9	10

Table 8.5: Overall feedback results summary

8.3.3 Research Questions

Using the score summaries presented in Figures 8.6, this subsection summarizes the results.

1. How can the visualizations help novice open-source developers

Question.	A_{answer}	B_{answer}
How many years of experience in software application design/ development or in any other activity do you have?	5 - 10 years	5 - 10 years
Have you ever contributed to a Open Source Development?	Yes	Yes
If you have selected Yes in the above question please list the projects that you have worked on	Shifu ml	XIO and Wala

Table 8.6: User background summary

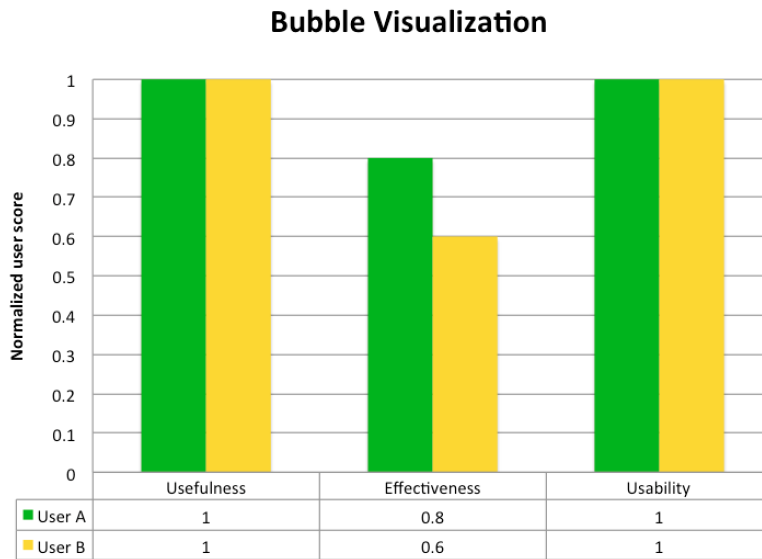


Figure 8.2: Bubble visualization results

and maintainers gain insights into cross-project porting activities?

The usefulness of visualizations presented in VIGNETTE can be inter-

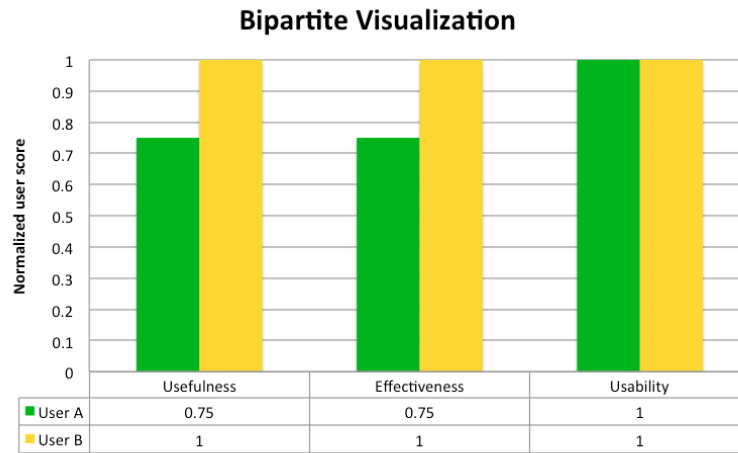


Figure 8.3: Bipartite visualization results

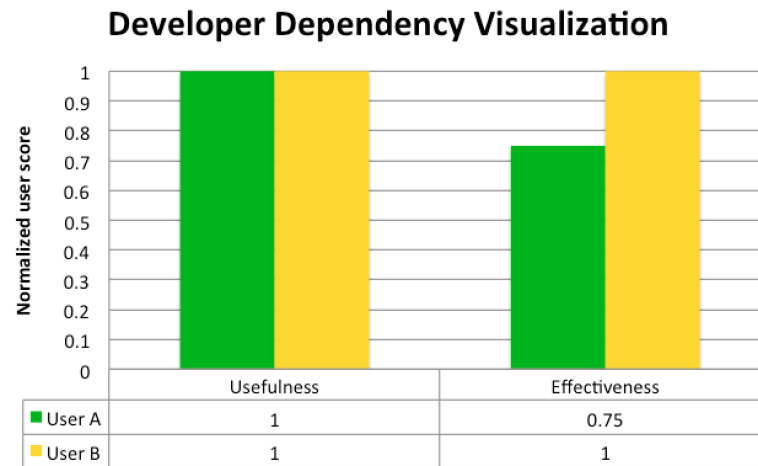


Figure 8.4: Developer dependency visualization results

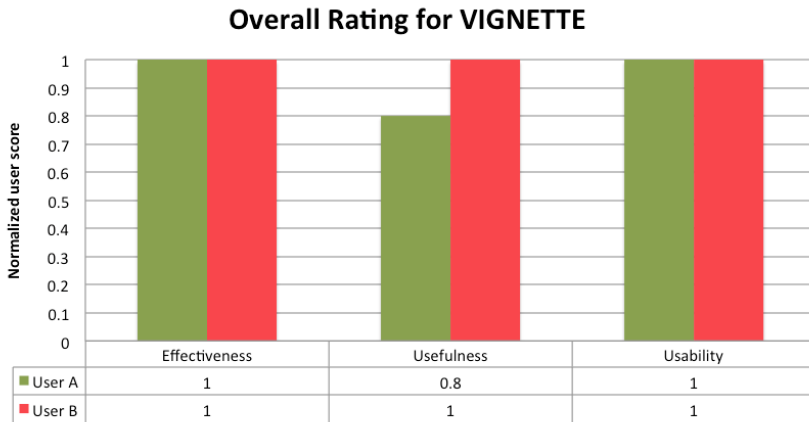


Figure 8.5: Overall results for VIGNETTE

Measurement Factors	User A	User B	Average	Percentage
Tree Visualization				
Usefulness	0.8	1	0.9	90
Effectiveness	0.8	1	0.9	90
Usability	0.67	1	0.835	83.5
Bubble Visualization				
Usefulness	1	1	1	100
Effectiveness	0.8	0.6	0.7	70
Usability	1	1	1	100
Bipartite Visualization				
Usefulness	0.75	1	0.875	87.5
Effectiveness	0.75	1	0.875	87.5
Usability	1	1	1	100
Dependency Wheel Visualization				
Usefulness	1	1	1	100
Effectiveness	0.75	1	0.875	87.5

Figure 8.6: Summary of normalized scores

preted from the Tables (8.1 to 8.4). The following is a brief summary of the conclusions: (1) The visualizations presented in this thesis can be of much value for performing porting activities effectively, (2) we found

that all visualizations could be highly useful for porting activities, and (3) the amount of information presented in the visualizations could be very useful for porting activities.

2. How well can VIGNETTE show the following based on cross-project porting activities?

(a) A file level association.

To understand the usefulness of file-level association for FreeBSD and NetBSD, we can consider the tree visualization results from Figure 8.6. The tree view could be very useful in identifying the file level association between FreeBSD and NetBSD projects and it is very easy to use.

(b) The pairwise frequency of porting.

We evaluate the pairwise frequencies among the three BSD projects using the bubble visualization ratings from Figure 8.6. The porting frequencies could be useful and easily identified from the visualizations. It could help a developer perform the porting activities effectively.

(c) The patch-file association.

To understand the impacts of visualization on patch-file relationships identification, we considered bipartite visualization results

from Figure 8.6. The bipartite visualization could effectively identify the porting activities and very easy to identify porting activities. Moreover, during the user study session, one user commented,

“Bipartite visualization is extremely useful in identifying porting activities based on the patch-file relationship.

I really like the details presented in this visualization.”

(d) **The developer to developer association.**

To analyze the usefulness of visualization in obtaining the cross-project developer association, we consider the dependency wheel visualization results in Figure 8.6. The dependency wheel visualization is very effective in identifying the developer association effectively.

Additionally, during the user study session, one participant made the following comment:

“The tool was extremely well built.

I really liked the idea of visualizing the raw data.”

Both users were very interested in knowing about VIGNETTE’s implementation details and asked the following questions:

“How did you implement the tool?

What technologies have you used?”

Based on the above analysis, we conclude that VIGNETTE is a useful tool for understanding porting activities. To completely reach a conclusion on the usefulness of this tool, further evaluation should be performed.

Chapter 9

Conclusion and Future Work

This chapter summarizes the thesis work. Section 9.1 presents the Threats to Validity. Sections 9.2 and 9.3 describe the summary and future work respectively.

9.1 Threats To Validity

This section describes the threats to construct, internal, external and conclusion validities.

9.1.1 Construct Validity

Threats to construct validity measure how well a study measures its claim. This thesis relied on cross-project ported edits identified by REPERTOIRE [18]. In the current implementation of the web-based tool, the results of porting analysis displayed as visualizations have certain restrictions. Tree visualization currently presents porting activities based on file name similarity for only FreeBSD-NetBSD project pairs for the entire project history. In order to display the visualizations at a web-page scale, bipartite visualization displays the porting activities for the top 20 patch IDs for FreeBSD and

NetBSD projects; meanwhile, the developer dependency visualization presents the relationships of 50 developers from both FreeBSD - NetBSD project pairs.

9.1.2 Internal validity

Threats to internal validity measure the accuracy of cause-effect inferences or causal relationships. This tool relies on REPERTOIRE [18] for the ported edits as inputs. All REPERTOIRE [18] threats to internal validity are applicable to VIGNETTE.

9.1.3 External Validity

Threats to external validity are a measure of the findings' generalizability. This study focuses on analyzing the porting activities in FreeBSD, NetBSD and OpenBSD. This tool may not generalize to other open-source projects where there is no evidence of cross-project porting activities and forking.

9.1.4 Conclusion Validity

Conclusion validity measures whether the study conclusions are reasonable. For the purpose of this thesis, we evaluated VIGNETTE with an exploratory user study using a group size of two members. As a result, the tool's usability measure is restricted to the level of expertise and experience of the two participants. Moreover, based on the given evaluation results, the study may or may not be applicable to a broad scope of open-source developers.

9.2 Summary

To analyze cross-project porting activities awareness, we have developed a web-based visualization tool, VIGNETTE. To support the geographically distributed community of open-source developers, we have chosen a web platform to implement the tool. VIGNETTE visualizes and presents the cross-project activity results as four visualizations. The user study results reveal that VIGNETTE can be very useful when learning about cross-project porting activities.

9.3 Future Work

In the future, we plan to improve the following aspects of the study.

Future Extension. As a future extension, we plan to extend the tool to support porting activities in other BSD projects, such as DragonFly BSD, and to support visualizations for different OSS projects, such as Mozilla and Linux, not only for porting activities but also for other maintenance activities.

Improve the Scalability of the Visualizations. In the current version of the tool, some of the visualizations (bipartite and developer dependency) have been designed to display only a few analysis results. For future extension, we plan to visualize all analysis results.

Recommendations of Ports. We plan to implement a functionality that automatically recommends ports to developers based on existing analysis results.

Appendices

Appendix A

User Study Reference Manual

A.1 Introduction

Maintaining multiple variants of a project is a time-consuming process, as new requirements and issues must be adapted to other project variants in a timely manner. *Software porting* is one of the common techniques employed to maintain forked open-source project variants; the developer manually ports feature enhancements from closely related projects. These activities are clearly evident in the family of BSD projects (FreeBSD, NetBSD, and OpenBSD). Because three projects evolved from a single BSD project, a bug fix in one project may affect some part of the code in other peer projects. Therefore, a developer of other peer projects may need to be aware of bug fix patches. This thesis hypothesizes that understanding cross-project porting dependencies is key to evolving project variants. To support the main objectives of this work, we designed a new web-based tool, VIGNETTE. The main goal of this study is to analyze the benefits associated with the web visualization, which aims to improve awareness of cross-project activities. We have designed the user study to introduce the tool's features, including a 30-minute tool demonstration.

A.1.1 Web-based Visualization of Cross-Project Porting

This work supports design and implementation of the following four different views to help developers understand porting activities: (1) tree view, (2) bubbled view, (3) bipartite view, and (4) dependency view. The following is a brief summary of each of these views.

1. **Tree** visualization shows a file level association of cross-project porting activities. The file level association can be defined as follows. For instance, if a user is applying a patch, say P1, and modifies a NetBSD file. If P1 is adapted to a file with the same name in OpenBSD, then the two files are associated. Developers can benefit from knowing the file-level associations between peer projects during a bug fix or feature enhancement. They can quickly learn more about the activities simply by clicking on a particular tree node rather than spending a great deal of time on such activities.
2. **Bubble** visualization gives the frequency of porting activities over the entire history of BSD projects. The frequency is represented as bubbles/circles of varying sizes and colors. With this view, users can quickly understand the frequency of cross-project porting activities.
3. **Bipartite** visualization presents a patch-file relationship constructed with two stacked bar charts displayed in a row to represent the patch-file relationship. Developers can learn about the number of times that the same patch was ported to a module in a peer project and a different

module of the same project. Developers can take advantage of the view to learn the location of dependent projects, in which an updated patch needs to be applied across different modules in the same project or in peer projects.

4. **Developer Dependency** visualization displays developer dependencies across closely-related projects. Using this visualization, the developers can obtain a list of cross-project developers to whom important information about porting activities needs to be communicated.

A.1.2 Study Overview

We have designed the study to be completed within approximately one hour. The study participants will be guided in exploring the web visualization tools by performing various tasks associated with the four different visualizations. The user study will begin with a 30-minute introductory session that includes a demo of the tool, followed by an exploratory task for each view. We have designed each task to be completed within approximately 10 minutes, including a feedback survey after each task. The user study will end with a post-questionnaire that can be completed within 5 minutes.

A.1.3 Demo Agenda

A demo of the tool includes a demonstration of the following.

- Overview of all the pages.

- Navigation between the pages.
- Showing all the features of the tools.
- Presenting the interactive features of the tools.

A.1.4 Tasks

Tree Visualization Tasks: explore the visualization by following the steps provided and answer the associated questions. This task is divided into two sub-tasks. The following is a brief overview of the tree view.

- Each **node** represents a file in the FreeBSD or NetBSD project.
- The **edge** represents the hierarchy of the projects.
- Each node can be clicked for further investigation.

Task 1.1

Please identify the affected files in NetBSD project by a patch that was applied to the module kern of the FreeBSD project for the year 2005.

Steps

1. Hit the below application URL in any browser.

<http://vignettetool.com/TreeVisualization/>

2. Select the Tree Visualization view from the home page.
3. Select the year **2005** from **Select Year** dropdown.
4. Select **FreeBSD** and **NetBSD** as the projects from **BSD Project 1** and **BSD Project 2** dropdowns.
5. Click the node named **kern** under the FreeBSD frame section → Pop window gets displayed → Select the **Cancel** button.
6. Observe updated tree of files in the **NetBSD frame**, this tree structure of files are files affected in NetBSD for the selected porting activity in freeBSD.

Questionnaire

1. **How would you rate Tree visualization in terms of presenting the file level porting association across FreeBSD and NetBSD projects?**
 - (a) Excellent
 - (b) Very Good
 - (c) Good
 - (d) Fair
 - (e) Needs Improvement

2. **In a role of a novice open source developer, would you like to use this tool during project development?**
 - (a) Would definitely use the tool.
 - (b) Would think about using the tool.
 - (c) Would prefer not to use.
 - (d) Unsure about the tool usage.

3. **Please suggest improvements for the tool,(if any) ———**

Task 1.2

Please investigate further to find out all the developers involved, their emails, commit dates, and patch IDs related to porting activities of the log module for the year 2005.

Steps

1. Hit the below application URL in any browser.
<http://vignettetool.com/TreeVisualization/>
2. Select the Tree Visualization view from the home page.
3. Select the year **2005** from **Select Year** dropdown.
4. Select **FreeBSD** and **NetBSD** as the projects from **BSD Project 1** and **BSD Project 2** dropdowns.

5. Click the node named **log** under the FreeBSD frame section → Pop window gets displayed → Select the **OK** button.
6. Observe and learn more about the additional information such as developer(s) involved, the Patch IDs and commit dates for the selected year of porting.

Questionnaire

1. **How would you rate the information provided on the additional details page for learning about porting activity?**
 - (a) Extremely useful
 - (b) Useful
 - (c) Additional information required
2. **If you have selected option c above, please suggest what information could be included on the page——**

Bubble Visualization Task: could be useful to gain insights about the frequency of porting throughout the entire history of the BSD project, to learn about the importance of porting, and to answer questions at the end of each task.

- The degree of frequency is represented as a colored bubble for a specific project.

- A bigger bubble represents the large porting activity.
- Each project in a source of porting can be identified with a respective color code.

Coordinates

- **X - Axis** - represents the year of porting.
- **Y - Axis** - represents the number of porting activities.

Additional functionality

The bubble chart was designed to handle hover events and mouse click events.

Task 2.1

Please identify the largest porting frequency for NetBSD-OpenBSD project pairs from the entire history of the BSD project.

Steps

1. Hit the below application URL in any browser.
<http://vignettetool.com/TreeVisualization/>
2. Select the Bubble Visualization view from the home page.
3. Now hover on the **NetBSD-OpenBSD** porting frequency bubble with largest size.

4. Next click on the largest **NetBSD-OpenBSD** bubble.
5. A pop-up gets displayed → Select the **OK** button → Additional information page is displayed.
6. Observe and learn more about developers involved, Patch Ids and commit dates for the selected year of porting.

Questionnaire

1. **Could you please select the year that has the largest porting frequency for NetBSD-OpenBSD project pair?**
 - (a) 1998
 - (b) 1999
 - (c) 2000
 - (d) 2001
 - (e) 2002
2. **After exploring the bubble chart, how much importance needs to be weighed on learning the porting activities?**
 - (a) Not at all important
 - (b) Little importance
 - (c) Important
 - (d) Highly important

3. On a scale of 1 to 5, how would you think the details provided in the additional details page could be utilized for further investigation?
- (a) 1 (Low)
 - (b) 2
 - (c) 3
 - (d) 4
 - (e) 5 (High)

Answer to (1): 2001

Bipartite Visualization Tasks is designed to offer a comparison view that displays the cross-project patch file relationship and the same project patch-file relationship. The main purpose of this task is to explore the patch-file relationship for a particular patch applied to different modules of the same project and across the peer project.

- Two stacked bar graphs are displayed side by side in each comparison view.
- The stacked bar graphs are connected by means of translucent lines indicating the patch file relationship.
- The weight of the bars varies based on the number of times that the same patch was used in different porting activities.

Coordinates

- The text on the left side of stacked bars represents the patch ID and the number of times the patch was used in different porting activities.
- The text on the right side of stacked bars represents the file count of the number of times that the file has ported from different patches.
- Color-coding of the bars is displayed according to the amount of porting.

Additional functionality Hover and click functionalities were integrated into this visualization.

Task 3.1

Please identify the different **FreeBSD** patch IDs that has been ported a single **NetBSD** file.

Steps

1. Hit the below application URL in any browser.
<http://vignettetool.com/TreeVisualization/>
2. Select the Bipartite Visualization view from the home page.
3. From **FreeBSD Patch to NetBSD** files (stacked bar view in the left section of the page) → Identify a **NetBSD file** that had two different **FreeBSD patch IDs** ported.

1. **Select the NetBSD file name that has two different Free Patch Ids applied?**

- (a) linux_sigaction.c
- (b) lx_oldmmap.c
- (c) ad1848.c
- (d) sys_generic.c

Answer: lx_oldmmap.c

Task 3.2

Please identify the FreeBSD patch ID that has been ported many times to the same FreeBSD files.

Steps

1. Hit the below application URL in any browser.
<http://vignettetool.com/TreeVisualization/>
2. Select the Bipartite Visualization view from the home page.
3. From **FreeBSD Patch to FreeBSD** files (stacked bar view in the left section of the page) → Identify a **FreeBSD file** that had two different **FreeBSD patch IDs** ported.

4. Click on the identified file name → A pop window gets displayed → Select the **OK** button → Developer Dependency wheel view is displayed.

1. **Select the FreeBSD file name that has two different Free Patch Id?s applied?**

- (a) ad1848.c
- (b) nic5000.c
- (c) pcvt_vtf.c
- (d) sys_generic.c

Answer: pcvt_vtf.c

Questionnaire

1. **Rate the ease with which the largest patch-file relationship could be identified?**

- (a) Very easy
- (b) Easy
- (c) Difficult
- (d) Very difficult

2. **After exploring the bubble chart, how much importance needs to be weighed on the learning of porting activities?**

- (a) Very easy
- (b) Easy
- (c) Difficult
- (d) Very difficult

3. **Who would benefit most from such a visualization type? (Select all that apply.)**

- (a) Manager
- (b) Developer
- (c) Committer
- (d) Maintainer
- (e) None

Developer Dependency Visualization Task: The main purpose of this view is to analyze and learn about the developer pair across the peer projects based on porting activities across the projects.

- The chart is represented as a Wheel of Developer Dependencies.
- Each developer is represented as an arc of the wheel chart.
- The developer-to-developer pair dependency is represented in the form of connecting chords.

- The width of the chord represents the number of times that the same developer pair was observed together in a porting activity.
- Each developer is represented with a distinct color.

Additional functionality

- Hovering over the developers name indicates the pairing percentage of the developer.
- Hovering over the chord connecting the developer pair gives additional information about the pairing.

Task 4.1

Identification of the two developers who were paired for the maximum number of times.

Steps

1. Hit the below application URL in any browser.
<http://vignettetool.com/TreeVisualization/>
2. Select the Wheeled Visualization view from the home page.
3. Identify the two developers who were paired with more number of peer project developers.

4. Hover on the developer with name **FBsd/dfr** (in Light red color) → Observe how active (as a percentage) the FreeBSD developer has been active from the tool tip.
5. Hover on the developers developer with name **FBsd/dfr** (in Light red color) → Observe the pair percentage for the developers in the tool tip.

Questionnaire

1. **How would you rate the amount of information presented on the peer project developer dependencies?**
 - (a) Extremely informative
 - (b) Highly Informative
 - (c) Informative
 - (d) Not Informative
2. **How would you rate the Developer Dependency Wheel in terms of knowing the list of affected developers while a patch is applied to a paired developer's work?**
 - (a) Extremely helpful
 - (b) Very helpful
 - (c) Not very helpful
 - (d) Not at all helpful

3. **What would you like to see improved in this view? Please make suggestions.**

A.1.5 Feedback Questionnaire

This section contains questions that gauge the participant's overall user experience along with questions that collect the user's background information.

1. **How would you rate the overall user experience (Look and Feel) of the web tool?**

- (a) 5 (Excellent)
- (b) 4 (Very Good)
- (c) 3 (Good)
- (d) 2 (Fair)
- (e) 1 (Poor)

2. **Is visualization a powerful medium for learning such porting activities? (Please rate the following on a scale of 1 - 5.)**

- (a) 5 (Extremely powerful)
- (b) 4 (Highly powerful)
- (c) 3 (Powerful)
- (d) 2 (Less powerful)
- (e) 1 (Not at all powerful)

3. Select the best/most useful visualization among the four visualizations of the web application?

- (a) Tree Visualization
- (b) Bubble Visualization
- (c) Bipartite Visualization
- (d) Dependency Wheel Visualization

4. How would you rate the ease of use (navigation between pages) of the web application?

- (a) 5 (Extremely easy)
- (b) 4 (Very easy)
- (c) 3 (Easy)
- (d) 2 (Not easy)
- (e) 1 (Not at all easy)

Background

1. How many years of experience in software application design/development or in any other activity do you have?

- (a) Less than 5 years
- (b) 5 - 10 years
- (c) 10 - 15 years

(d) 15 - 20 years

(e) More than 20 years

2. Have you ever contributed to an open source development?

(a) Yes

(b) No

3. If you selected Yes in the question listed above, please list the projects that you have worked on ———-

Bibliography

- [1] Eytan Adar and Miryung Kim. Softguess: Visualization and exploration of code clones in context. In *ICSE*, volume 7, pages 762–766, 2007.
- [2] Mihai Balint, Tudor Girba, and Radu Marinescu. How developers copy. In *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on*, pages 56–68. IEEE, 2006.
- [3] Jacob T Biehl, Mary Czerwinski, Greg Smith, and George G Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1313–1322. ACM, 2007.
- [4] Gerardo Canfora, Luigi Cerulo, Marta Cimitile, and Massimiliano Di Penta. Social interactions around cross-system bug fixings: the case of freebsd and openbsd. In *Proceedings of the 8th working conference on mining software repositories*, pages 143–152. ACM, 2011.
- [5] Kenneth J Chan and David Jackson. Porting under unix-problem areas and a proposed strategy. In *PROCEEDINGS EUUG CONFERENCE*, pages 15–15, 1992.
- [6] Marco D’Ambros and Michele Lanza. A flexible framework to support collaborative software evolution analysis. In *Software Maintenance and*

- Reengineering, 2008. CSMR 2008. 12th European Conference on*, pages 3–12. IEEE, 2008.
- [7] Marco D’Ambros and Michele Lanza. Visual software evolution reconstruction. *Journal of Software Maintenance and Evolution: Research and Practice*, 21(3):217–232, 2009.
- [8] Massimiliano Di Penta, Daniel M German, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An exploratory study of the evolution of software licensing. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 145–154. ACM, 2010.
- [9] Ekwa Duala-Ekoko and Martin P Robillard. Tracking code clones in evolving software. In *Proceedings of the 29th international conference on Software Engineering*, pages 158–167. IEEE Computer Society, 2007.
- [10] Michael Fischer, Johann Oberleitner, Jacek Ratzinger, and Harald Gall. *Mining evolution data of a product family*, volume 30. ACM, 2005.
- [11] Daniel M German, Massimiliano Di Penta, Y-G Guéhéneuc, and Giuliano Antoniol. Code siblings: Technical and legal implications of copying code between applications. In *Mining Software Repositories, 2009. MSR’09. 6th IEEE International Working Conference on*, pages 81–90. IEEE, 2009.
- [12] Cory Kapser and Michael W Godfrey. Improved tool support for the investigation of duplication in software. In *Software Maintenance, 2005*.

- ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 305–314. IEEE, 2005.
- [13] Rainer Koschke. *Survey of research on software clones*. Internat. Begegnungs-und Forschungszentrum für Informatik, 2007.
- [14] Jens Krinke. Identifying similar code with program dependence graphs. In *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, pages 301–309. IEEE, 2001.
- [15] Michele Lanza. Codecrawler-lessons learned in building a software visualization tool. In *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, pages 409–418. IEEE, 2003.
- [16] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *Software Engineering, IEEE Transactions on*, 32(3):176–192, 2006.
- [17] Masao Ohira, Naoki Ohsugi, Tetsuya Ohoka, and Ken-ichi Matsumoto. Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.
- [18] Baishakhi Ray and Miryung Kim. A case study of cross-system porting in forked projects. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 53. ACM, 2012.

- [19] Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 1999.
- [20] Eric S Raymond. Homesteading the noosphere. *First Monday*, 3(10), 1998.
- [21] Matthias Rieger and Stéphane Ducasse. Visual detection of duplicated code. In *ECOOP Workshops*, pages 75–76. Citeseer, 1998.
- [22] Robert Tairas, Jeff Gray, and Ira Baxter. Visualization of clone detection results. In *Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*, pages 50–54. ACM, 2006.
- [23] David M Tilbrook and Russell Crook. Large scale porting through parameterization. In *USENIX Summer*, 1992.
- [24] Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Gemini: Maintenance support environment based on code clone analysis. In *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, pages 67–76. IEEE, 2002.
- [25] Patrick Wagstrom, Corey Jergensen, and Anita Sarma. A network of rails a graph dataset of ruby on rails and associated projects. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pages 229–232. IEEE, 2013.
- [26] David A Wheeler. Why open source software/free software (oss/fs, floss, or foss)? look at the numbers! 2005.

- [27] Robert Young. Giving it away: How red hat software stumbled across a new economic model and helped improve an industry. *Journal of Electronic Publishing*, 4(3), 1999.
- [28] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 91–100. ACM, 2009.

Vita

Yamini Gotimukul is a M.S.E. student in the Department of Electrical and Computer Engineering at the University of Texas at Austin. Her research focuses on software engineering, in particular, visualization analysis and tools for evolving open source software. Yamini was born in Hyderabad, India. She earned her bachelor's degree in 2008 from Osmania University, with Electronics and Communication as major. She worked as a programmer analyst in Cognizant Technology Solutions and as a web developer in Yana Software Inc.

Email address: yamini.gotimukul@utexas.edu

This thesis was typeset with L^AT_EX[†] by the author.

[†]L^AT_EX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's T_EX Program.