VerXCombo: An interactive data visualization of popular library version combinations

Yuki Yano, Raula Gaikovina Kula, Takashi Ishio, Katsuro Inoue
Osaka University, Japan
{y-yano, raula-k, ishio, inoue}@ist.osaka-u.ac.jp

***Abstract*—In large software systems, it is common practice to adopt third-party libraries. Decisions by system maintainers to either update or introduce new third-party libraries can range from trivial to complex. For instance, incompatibility between internal library dependencies may complicate adoption. Therefore, system maintainers especially need adequate assurance of any candidate library release. Using the *'wisdom of the crowd'*, VerXCombo aims to assist system maintainers by mining popular library dependency patterns of similar systems. Through data interactions, VerXCombo leverages parallel sets to break-down large and complex dataset into distinguishable patterns of 1.) popular and 2.) latest library dependency release combinations. Populating our tool with a maven library dependency dataset from over 4,000 Java Open Source projects, we demonstrate through a case scenario navigation and best fit combinations of the VerXCombo tool. A video highlighting the main features of the tool can be found at: http://goo.gl/wWPylL**

***Index Terms*—Software reuse, Mining software repository, Software Visualization**

I. Introduction

In software development, the adoption of third-party software libraries is becoming common-place. Benefits include reducing time and effort costs of *'reinventing the wheel'*, with the added assurance of quality. With the expansion of library hosting sites such as the Maven 2 Central Repository[1] for Java and RubyGems[2], system maintainers now have open access to a vast range of Open Source System (OSS) libraries. OSS third-party library adoption have become widespread. Usage in both open and industrial settings has validated OSS library usage[1].

As a library evolves, maintenance issues arise as whether to incorporate new bug fixes and recent improvements and features. As part of software maintenance, system maintainers need to consider *'if'* and *'when'* existing libraries should be updated to *'which'* library version. Moreover, system maintainers are usually faced with introducing a new library. In this case, they need to consider how this new library will best *'fit-in'* with the existing dependency environment. Studies have expressed concerns of incompatibility when updating [2], [3], while examples such as the heartbleed bug[3] illustrate the widespread effect of when updates are disregarded. These tasks can be become extremely challenging when documentation of the system is poor or knowledge of the system is lost due to project personnel changes.

[1]http://search.maven.org/
[2]https://rubygems.org
[3]http://heartbleed.com/

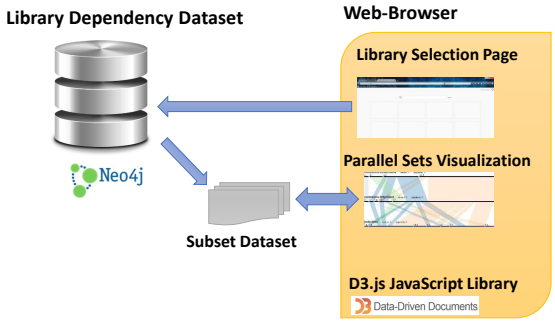

Fig. 1. An architectural overview of VerXCombo

With the rise of OSS and its open access repositories, mining techniques have been employed to gather useful significant amounts data on patterns of library usage. However, the data relationships are often complex and too large to properly visualize and navigate.

To this end, we propose VerXCombo (Version X Combination) as a prototype to present various combinations of library versions. Using the 'wisdom of the crowd' popularity metrics, VerXCombo is able to depict different combination patterns that allow the users to determine the best-fit combination.

II. VerXCombo Overview

A. Architectural Design

VerXCombo is built as a platform independent web application, with an easy–to–use interface. It is implemented with HTML5 and JavaScript for the front-end and Apache Tomcat[4] and a Neo4j[5] graph database for the server backend.

Figure 1 shows an architectural overview of VerXCombo. The library dependency datasets are stored in the Neo4j VerxCombo database. Through the web-interface, the user initially selects a set of libraries that exist in the database. To enable the data interaction, a subset dataset containing only information specific to the query is returned.

From the returned subset dataset, VerXCombo employs the d3.js[6] JavaScript engine to interact with the manageable subset dataset. Displayed as parallel sets, users are then able to sort

[4]http://tomcat.apache.org/
[5]http://neo4j.com
[6]http://d3js.org/

978-1-4673-8159-8/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPC.2015.43

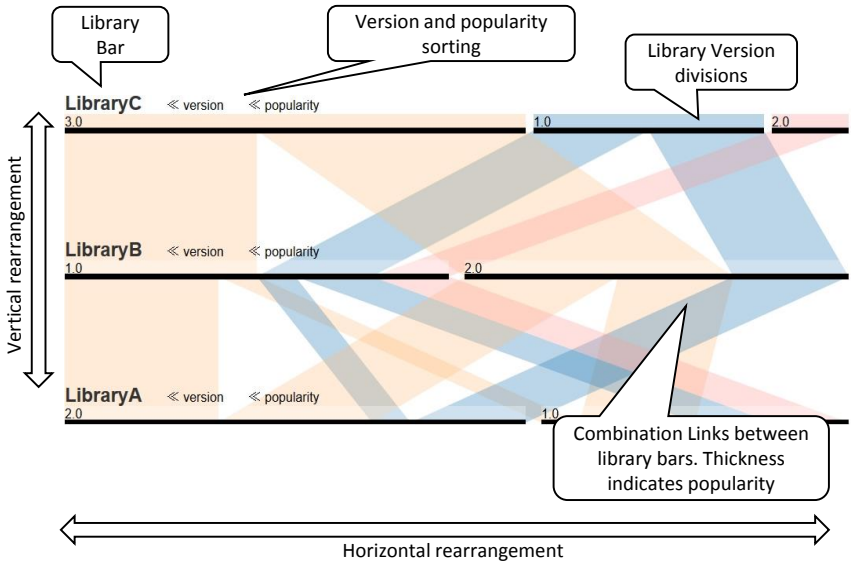

Fig. 2. VerXCombo - Parallel Sets Visualization

and order different library combinations according to the 1.) popularity and 2.) latest release sets.

B. Library Combinations as Parallel Sets

Parallel sets serve as an interaction framework and visual metaphor to naturally map categorical variables to visual entities [4]. As an extension of the bipartite graph, parallel sets are an interactive exploration of categorical data that shows data frequencies instead of the individual data points.

Figure 2 depicts a simplified example of the parallel set interactions. Each of the three parallel bars represent a library, with the target library located on the bottom bar. The bar then is divided into each respective library versions.

A combination link between the library bars depicts the frequency count of systems that uses both library versions. The thickness of the link is the propositional frequency of systems that use the corresponding libraries.The color corresponds to a version of the target library.

As seen in Figure 2 $\text{LIBRARYC}_{1.0}$ has a light blue colored combination link to $\text{LIBRARYB}_{1.0}$ and $\text{LIBRARYB}_{2.0}$. Fundamental interaction is performed by either the vertical rearrangement of the library bars or horizontal rearrangement of the versions on the bar of the library. The mouseover event highlights a particular combination set, also providing statistics on this combination set.

The vertical rearrangement enables users to focus on direct links in relation to the target library. The horizontal rearrangement of the library versions can be manipulated to show the order of popularity (most popular to least popular usage) or the sequence of release times (oldest to newest).

C. Best-Fit: Popularity and Latest Version Release

We recognize the following assumptions as objectives to determine the 'best-fit' combination of library versions:

- *Popularity use*. We assume that popular use by similar system indicates a favorable library version to adopt.
- *Latest Version*. We assume that system maintainers would like to keep 'up-to-date' with the current bug fixes and enjoy latest features of a library.

We leverage the interactive and frequency features of VerXCombo to determine popularity the latest version release 'best-fit' combinations. Visually, the thickness of a combination sets indicates its popularity. Figure 2 shows that VerXCombo has both popularity and version horizontal sorting settings. Users can interact with the visualization to determine best-fit.

III. Illustrative Usage Scenario

We demonstrate the usefulness of VerXCombo in a realistic environment setting. Using our research prototype, we perform a use case scenario in respect to library update decisions.

A. Library Dependency Dataset

For this scenario, we populated the VerXCombo database with systems that depend on java libraries that are managed and hosted on the Maven 2 Super Repository. For a representative sample, we analyzed library dependency information from 4,367 projects hosted on GitHub[7]. We used an extension of our Pomwalker tool[8] to extract to system and library dependencies from their respective `pom.xml` files. Table I provides details of the dataset we used to populate the VerXCombo database.

B. Real-world Scenario: Introducing a new library

System S_x is part of a web-based application, that has existing dependencies with the Apache $\text{COMMONS-COLLECTIONS}_{verX}$ and $\text{COMMONS-HTTPCLIENT}_{verX}$.

[7]https://github.com
[8]https://github.com/raux/PomWalker

TABLE II
CASE STUDY SCENARIO - ALL LIBRARIES ARE EXTENSIONS OF THE STANDARD JAVA SDK LIBRARY.

Library	Description	System Environment Version	Latest Version
COMMONS-COLLECTIONS	Java Utilities Library (java.util extension)	3.2	3.2.1
HTTPCLIENT	Client-side HTTP protocol library (java.net extension)	3.1	3.1
JODA-TIME	Date, Time and Calender library (java.time extension)	-	2.3

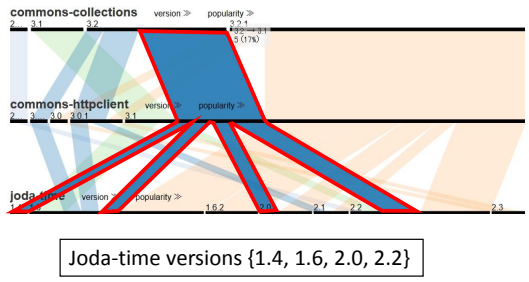

Fig. 3. Joda-time - Possible Combinations

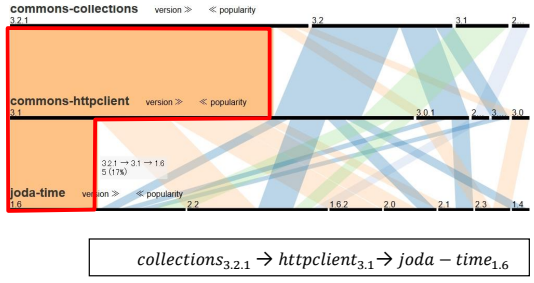

Fig. 4. Joda-time - Popular Combination

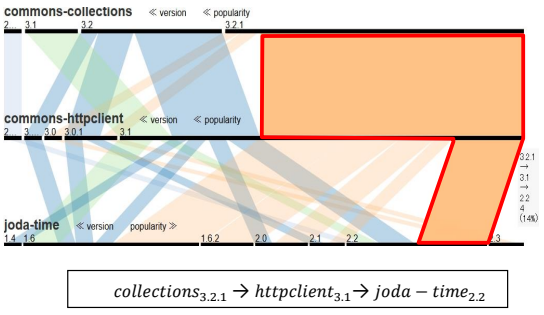

Fig. 5. Joda-time - Latest Version Combinations

TABLE I
LIBRARY DEPENDENCY DATASET

# of Projects	4,210
# of Systems	36,910
# of Libraries	29,535
# of Libraries Versions	151,647

COMMONS-COLLECTIONS is an expansion of the JDK utility, with newer interfaces, implementations and utilities[9]. HTTPCLIENT extends on the java.net library by providing an efficient, up-to-date, and feature-rich package implementing the client side of the most recent HTTP standards and recommendations[10].

In the next system release (i.e. S_{x+1}), the system maintainer would like to implement a new functionality that requires replacing the standard Java date and time classes. The system maintainer would like to adopt the more powerful JODA-TIME[11] library as a target candidate. Using VerXCombo, the system maintainer can now explore the most suitable library combinations between these three libraries.

Table II depicts the version information for the respective libraries. The latest library version is also included. Based on the scenario and the library dependency dataset provided, VerXCombo can be used to evaluate the following questions:

- Q1: What set of the target library version combinations 'fit-in' the current system dependency environment?
- Q2a: What is the most popular combination set of libraries?
- Q2b: What combination satisfies the closest to the latest version combination?
- Q3: Which combination is recommended to enable use of the latest target library version?

C. Findings

Table III provides a brief summary of the visual findings from the case scenario. In response to Q1, as shown in Figure 3, we find that JODA-TIME versions $\{[[1.4], 1.6], [2.0], [2.2]\}$ have been used by similar systems that match the current environment settings of COMMONS-COLLECTIONS$_{3.2}$,COMMONS-HTTPCLIENT$_{3.1}$. Using the mouseover event, we are able to highlight these versions. Figure 4 depicts the most popular combination set. Therefore to answer Q2a, the combination $\{$COMMONS-COLLECTIONS$_{3.2.1}$, COMMONS-HTTPCLIENT$_{3.1}$,

[9]http://commons.apache.org/proper/commons-collections/
[10]http://hc.apache.org/httpcomponents-client-ga/index.html
[11]http://www.joda.org/joda-time/

TABLE III
SCENARIO FINDINGS FROM VERXCOMBO

		VerXCombo Recommendation
Q1	Popular Target Versions	[JODA-TIME$_{[1.4,1.6,2.0,2.2]}$]
Q2a	Popular Fit	[COMMONS-COLLECTIONS$_{3.2.1}$, COMMONS-HTTPCLIENT$_{3.1}$, JODA-TIME$_{1.6}$]
Q2b	Latest Fit	[COMMONS-COLLECTIONS$_{3.2.1}$, COMMONS-HTTPCLIENT$_{3.1}$, JODA-TIME$_{2.2}$]
Q3	Latest Target	[COMMONS-COLLECTIONS$_{3.2.1}$, COMMONS-HTTPCLIENT$_{3.0.1}$, JODA-TIME$_{2.3}$]

JODA-TIME$_{1.6}$]} is the popular fit. Do note however, that JODA-TIME$_{1.6}$ is a much older version so may not be the best option.

Finding the fit with respect to the latest versions (Q2b) includes the combination of the most recent releases of all libraries. In this case, the combination {COMMONS-COLLECTIONS$_{3.2.1}$, COMMONS-HTTPCLIENT$_{3.1}$, JODA-TIME$_{2.2}$} is recommended as the best-fit. JODA-TIME$_{2.2}$ although not the latest closest to the latest version. This combination also recommends the update to COMMONS-COLLECTIONS$_{3.2.1}$. It is interesting to note that the latest version JODA-TIME$_{2.3}$ is used as a combination with an older version of COMMONS-HTTPCLIENT$_{3.0.1}$, suggesting a downgrade one of the current libraries. Finally for Q3, COMMONS-COLLECTIONS$_{3.2.1}$, COMMONS-HTTPCLIENT$_{3.0.1}$, JODA-TIME$_{2.3}$ combination is recommended for the latest version JODA-TIME$_{2.3}$.

Ultimately, the system maintainer can use the VerXCombo tool to make an informed decisions by exploring the different library version combinations. Our final recommendation is either JODA-TIME$_{2.2}$ or JODA-TIME$_{2.3}$ with the option to update COMMONS-COLLECTIONS$_{3.2.1}$ without compromising any library version downgrades. Using the recommendation as a guide, system maintainers can proceed to look at documentation of the specifications to validate adoption.

D. Threats and Considerations

An ambitious system maintainer may decide to pioneer a new combination set of libraries. However, most developers may care about untested and early bugs or new releases. There are many other factors such as the compiler, development environment and programming language that may influence a system maintainers decision to update a library. In this work, we specifically target existing system dependency libraries to reduce the change of library incompatibility issues.

We demonstrate with a case scenario of maven libraries, however, we believe that the visualization can be extended to other programming languages and their library dependencies. Due to the large size of the dataset, we consider the data as a fair representation of typical OSS projects. The case scenario demonstrates that the tool is scalable to handle large datasets.

IV. RELATED WORK

There exists a large body of research that have studied library migrations and updates such as [5], [6]. All these studies highlight the challenges associated with library upgrades. Our tool provide developers with more information on specific libraries. Popularity measures have been used for API Usage [7], [8]. Work by Mileva and colleagues [9] visualized popularity trends of a single OSS library. A similar application of bi-partite sets has been proposed for Code Flows [10].

In our previous work [3], our results suggest that maintainers are more inclined to adopt the latest version releases. Complementary, our other visualizations [11] includes systems and their dependent libraries. In this work, we compare and explore different combinations of libraries.

V. SUMMARY & FUTURE WORK

We present VerXCombo as a tool to assist system maintainers make library maintenance decisions. We show through a real-world scenario some 'best-fit' result combination sets of libraries. Future work includes feedback and prototype use with real-world system maintainers.

ACKNOWLEDGMENT

This work is supported by 'SARF' Project, Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (No.25220003) and 'Software License Evolution Analysis', Osaka University Program for Promoting International Joint Research.

REFERENCES

[1] C. Ebert, "Open source software in industry," vol. 25, no. 3, May 2008, pp. 52–53.

[2] S. Raemaekers, A. van Deursen, and J. Visser, "Semantic versioning versus breaking changes: A study of the maven repository," in *Proc. of SCAM*, Sept 2014, pp. 215–224.

[3] R. G. Kula, D. M. German, T. Ishio, and K. Inoue, "Trusting a library: A study of the latency to adopt the latest maven release," in *22nd IEEE Int. Conf. on Soft. Ana., Evo., and Reeng., SANER 2015, Montreal, Canada, March 2-6, 2015*, 2015.

[4] F. Bendix, R. Kosara, and H. Hauser, "Parallel sets: Visual analysis of categorical data," in *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*. IEEE, 2005, pp. 133–140.

[5] C. Teyton, J.-R. Falleri, M. Palyart, and X. Blanc, "A study of library migrations in java," *Journl of Soft.: Evo. and Pro.*, vol. 26, no. 11, 2014.

[6] V. Bauer, L. Heinemann, and F. Deissenboeck, "A structured approach to assess third-party library usage," in *Software Maintenance (ICSM), 2012 28th IEEE Int. Conf. on*, Sept 2012, pp. 483–492.

[7] C. De Roover, R. Lämmel, and E. Pek, "Multi-dimensional exploration of api usage," in *Proc. of Int. Conf. on Prog. Comp.(ICPC)*, 2013.

[8] R. Holmes and R. J. Walker, "Informing Eclipse API production and consumption," in *OOPSLA2007*, 2007, pp. 70–74.

[9] Y. M. Mileva, V. Dallmeier, and A. Zeller, "Mining API popularity," in *TAIC PART*, 2010, pp. 173–180.

[10] A. Telea and D. Auber, "Code flows: Visualizing structural evolution of source code," *Computer Graphics Forum*, vol. 27, no. 3, pp. 831–838, 2008.

[11] R. G. Kula, C. De Roover, D. M. Germán, T. Ishio, and K. Inoue, "Visualizing the evolution of systems and their library dependencies," in *Second IEEE Work. Conf. on Soft. Vis., 2014, Victoria, BC, Canada, Sept. 29-30, 2014*, 2014, pp. 127–136.