González, A., Gómez, D. A., Segrera, S., Therón, R., Moreno, M. N., & García-Peñalvo, F. J. (2008). Uncovering the Relationships among Classes and Packages in Software Evolution. In M. N. Moreno García, A. B. Gil González, & V. F. López Batista (Eds.), Web Mining and Semantic Web. I Workshop, MiWebSe 2008 (27 October – 9 November, 2008) (pp. 11-20). Salamanca, Spain: Departamento de Informática y Automática de la Universidad de Salamanca.

Uncovering the relationships among classes and packages in software evolution

Antonio González, Diego A. Gómez, Saddys Segrera, Roberto Therón, María N. Moreno, and Francisco J. García

Department of Computer Science and Automatics,
University of Salamanca,
Plaza de la Merced s/n, 37008 Salamanca, Spain
{agtorres,dialgoag,saddys,theron,mmg,fgarcia}@usal.es
http://informatica.usal.es

Abstract. This paper proposes the uncovering and representation of the hierarchy and implementation relationships among software items using data stored in online software repositories. Software Configuration Management (SCM) tools and software repositories provide information about software item revisions and support evolution data extraction as well as the automatic retrieval of source code. Hence, the proposal presented here performs static analysis on the source code of software item revisions for collecting relevant data about relationships and dependencies among items. The visualization contributes to the comparison between revisions in order to provide a better understanding of structure changes on software projects.

Key words: Information retrieval, software repositories, information visualization, search engine

1 Introduction

Web mining is defined as the application of data mining techniques to discover and extract information from web documents and services automatically [1]. Nevertheless, data mining techniques are no easy of applying to web data due to problems related as much to the underlying technology as with the absence of standards in the design and implementation of Web pages.

Web mining consists of three tasks: source discovery, selection and information preprocessing, and general pattern discovery from web sites. This paper is aimed to the first of the Web mining tasks applied to online open source software repositories and the gathering of software evolution data. The information source discovery or the information retrieval (IR) consists of the automatic recovery of relevant documents, whereas it makes sure at the same time, as it is possible, that the non relevant ones are not considered.

At this point, it is relevant to consider that the use of online software repositories and SCM tools has become widespread in middle and large size software

projects. Besides, many commercial and non-commercial SCM and software versioning tools are available in the marketplace. According to [2] SCM enhances the environment of developers, managing concurrency and collaboration, and recording changes in software repositories; including time, date, which modules were affected, how long the modification took and information about who performed the change. SCM tools support many other functions, but the above functions are shared by most code versioning tools; such as Subversion [3].

Furthermore, online software repositories have been the information source for several researches in mining software repositories [4], and the visualization of the evolution of software projects and items [5, 6]. However, they had been ignored for a long time [7] despite their richness in providing valuable information on the detection of errors, pattern identification, structure changes and project management [8].

Moreover, the researches on IR admit modeling, user interface development, data visualization, and filters [9]. The Web content visualization is possibly the most complex part of the Web mining visualization. Not only by the vast and diverse contents of the web, but also by complexity of its semantics [10].

In recent years, the field of information visualization has played an important role in providing insight through visual representations combined with interaction techniques that take advantage of the human eye's broad bandwidth pathway to the mind, allowing experts to see, explore, and understand large amounts of information at once [11]. If the visualization takes place in an interface whose objective is information retrieval, the expression used for this type of system is Visual Information Retrieval Interfaces (VIRIs) [12].

The main contributions of this paper are the uncovering and representation of hierarchy and implementation relationships between Java software items, within the software project and external software items that belongs to the Java API or libraries used by the project under consideration. Our proposal considers the visualization of these relationships through several revisions for contributing to the comparison of structure changes on software projects.

This paper is structured in 5 sections, as follows: section 2 reviews related works, section 3 discusses our visualization design, section 4 presents a case study, and section 5 discusses the conclusions.

2 Related work

2.1 Information retrieval on the Web

Traditional information retrieval, which was developed for isolated databases, is not suitable for Web applications. The main differences between traditional and Web-based information retrieval are the number of simultaneous users of popular search engines, the number of documents which can be accessed [13] and technologies [14].

On the one hand, the number of simultaneous users of a search engine at an specific moment can not be predicted and it may overload a system and on the

other hand, the number of publicly accessible documents on the Web outperforms those quantities associated with traditional databases. In addition the number of Web search engine providers, Web users and Web pages have experienced a dizzy growth in the last years; an average page requires more space, uses more memory and involves more different types of multimedia information in relation to an average traditional document [13]. Furthermore, much of the Web information is semi-structured due to the nested structure of HTML code; it is linked and redundant. The Web is noisy, a Web page typically contains a mixture of many kinds of information such as main contents, advertisements, navigational panels, copyright notices [14].

Various techniques attempt to improve the search performance of Web pages by taking Web page structures into account, for example: giving more credit to words appearing in the title field, considering the distance between search keywords appearing within a page, assigning appropriate weights to each search keyword, or suggesting different paradigms (probabilistic, logic or language-based model) or formulas when computing the degree of similarity between a Web page and the user's query [9].

There are three Web-based information retrieval architectures: centralized [15], distributed [16, 15] and metasearch [17, 18, 15].

2.2 Software configuration management and mining software repositories

SCM tools and software versioning tools have been traditionally used by most software projects for providing a collaboration means between software developers. Nowadays, CVS [19] and Subversion [3] are the most widely used software versioning tools. According to the definition of the IEEE Standard 828-1990 [20] "Software configuration management (SCM) activities include the identification and establishment of baselines; the review, approval, and control of changes; the tracking and reporting of such changes; the audits and reviews of the evolving software product; and the control of interface documentation and project supplier SCM". Thus, SCM and software versioning tools use software repositories that provide data structures for storing and retrieving software items'revision changes. As a result, these tools can provide snapshots of the project code for each revision.

Some authors [21] support the idea that SCM repositories are valuable for project accounting, development audits and understanding the evolution of software projects. Furthermore, other authors [22] discuss about the challenges of retrieving and analyzing data automatically from software repositories. Additionally, the mining of software repositories have focused on specific dissimilarities between revisions and changes to artifacts on different granularity levels, such as classes and methods [4]. A framework for describing and understanding mining tools has been proposed in [23].

2.3 Information and software visualization

Information visualization deals with the representation and display of a large number of data elements, and provides visual representations to help the interpretation of a data element through its relation with other data elements. It takes under consideration several techniques to support navigation, interpretation of visual elements and understanding relationships among visual items in their full context [24]. Tufte states that the visual distinctions between visual elements should be as subtle as possible, yet clear and effective [25] adding that information consists of differences that make the difference [26].

A visualization for SCM repositories is proposed in [27]. That proposal demonstrated that the adequate use of 2D visualizations in conjunction with colors and textures contribute to the development of powerful multidimensional visualization solutions.

Finally, [28] present a visualization of a Dependency Structure Matrix for representing modules dependencies on software projects and [29] propose an evolutionary approach applied to the visualization of software item dependency.

3 Description of the proposal

This paper focuses on the mining of online software repositories and the visualization of software items dependency through several revisions. We consider the proposal presented by [30] that discuss the restructuring of hierarchical relationships between classes and defines four types of hierarchy relationships: self-to-self, external to external, self to external and external to self; where self refers to the current project and external to another project.

Hence, we have extended and redefined these definitions as self-to-self, library to library, Java API to Java API, self to library, self to Java API, library to Java API, Java API to library, library to self and Java API to self. Where self refers to the current project, library to an external library or another project and Java API to the Java API itself. Moreover, we have also considered the implementation of interfaces and classes, interfaces, enums and annotations as the software items that can participate of inheritance and implementation.

The information extraction about software items dependency is done for each revision of the project on a static based analysis. The SoftMiner tool locally stores each source code revision and then the Analyzer takes the class header and determines if its parent (in case it has one) belongs to the current project, an external library or the Java API. The analyzer follows a similar approach for the interfaces specified under the implements keyword; additional details can be reviewed on algorithm 3.1.

Algorithm 3.1: EXTRACTION OF SOFTWARE ITEM DEPENDENCY(*repos*)

```
repository ← repos
while n < repository.lastRevision()
do { revisionClasses ← SoftMiner.getRevisionClasses(n)
     for each class ∈ revisionClasses
     do { if class.hasParent()
          then { par ← class.getParent()
                 self ← Analyzer.isParentOnProj(par)
                 if self
                   then location ← "Self"
                   else java ← Analyzer.isParentOnJava(par)
                 if java
                   then location ← "Java"
                   else location ← "Library"
                 package ← Analyzer.getClassPackage(par)
                 extends(x) = (parent location package)
                 vector.add(extends)
          if class.hasImplements()
          then { interfaces ← class.getInterfaces()
                 for each int ∈ interfaces
                 do { self ← Analyzer.isIntOnProj(int)
                      if self
                        then location ← "Self"
                        else java ← Analyzer.isIntOnJava(int)
                      if java
                        then location ← "Java"
                        else location ← "Library"
                      package ← Analyzer.getIntPackage(int)
                      implements(x) = (parent location package)
                      vector.add(implements) } } } }
return (vector)
```


A description of the visualization is presented on figure 1. It shows three packages containing "i" and "c" ovals linked by black lines. Ovals labeled with "c" letters represent classes and ovals labeled with "i" letters represent interfaces and the small black oval in one of the line ends represents the linking to a parent class or to an interface. Furthermore, the size of ovals represents the number of relationships with other software items.

This representation uses ovals for the grouping of classes in the same package and graph of forces for placing closely classes that have and inheritance or implementation relationship. It is a valuable visualization because it represents the dependency relationship among software items and their packages, as well as the location of classes within the structure of the software project. The main limitation of this representation is that it only analyzes classes written in Java.

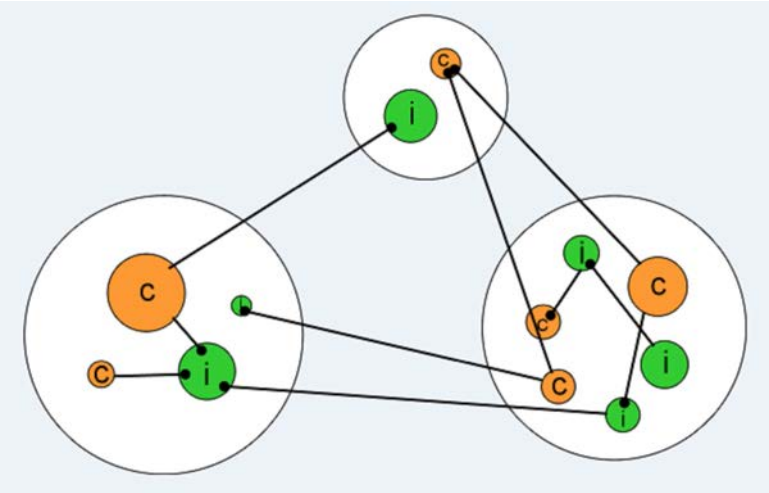

Fig. 1. Software item dependency design.

We support our visualization proposal through linked and focus+context views, selection, navigation, filtering and zoom interaction mechanisms, in addition to polyfocal display, a tree hierarchies and a timeline visualization technique.

4 Case study

This section discusses the application of the proposal to jEdit, an open source software project which started on December, 1999 and currently has nearly 612 classes and 3050 revisions.

We propose the following list of questions for evaluating our visual representation:

1. Items
   - 1.1 Which software item is the parent of other software items?
   - 1.2 Which interfaces do the software item implements?
   - 1.3 Does the software item inherits from a class or interface located on the same package, the same project, the Java API or an external library?
   - 1.4 Does the software item implements interfaces located on the same package, the same project, the Java API or an external library?
   - 1.5 How many levels of inheritance a software item has?
   - 1.6 Which software item has more relationships with other software items?
   - 1.7 What is the location of the software item?
2. Packages
   - 2.1 How many packages are part in the project?
   - 2.2 Which package has more relationships with other packages?
3. Tool

3.1 Does the tool provide zoom and selection?

When assessing the tool, see figure 2, with the above questions it is easy to realize after selecting a software item which are its children (question 1.1) as well as which interfaces it implements (question 1.2) . Additionally, it shows the relationship of a software item with other software items on the same package, the same project, the Java API or an external library (questions 1.3 and 1.4).

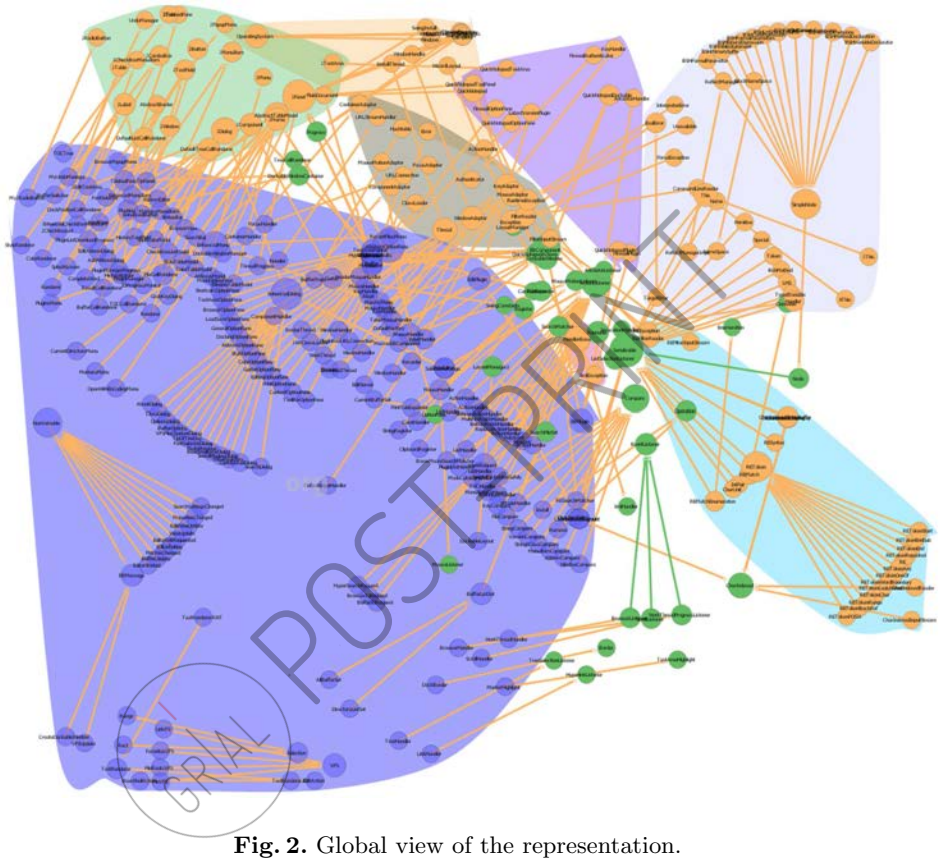

Fig. 2. Global view of the representation.

The representation proposal offers the possibility of disclosing the levels of inheritance of software items, consequently reveals the propagation of changes when software items on the top of the hierarchy change (question 1.5). Figure 3 shows the selection of interface Node and highlights its parent interface, the classes that implements it, and even the software items with an indirect relationship. This visualization also displays visual information about the relationships of a software item with others (question 1.6) and the package where a software item is located (question 1.7).

In addition, one can determine how many packages are in the project easily (question 2.1); it is only required to do some selection or zooming (question 3.1). Finally, the identification of which packages have more or less relationships with other packages is clear at first glance.

This visualization supports many additional features that help answer many more questions after a carefully visual inspection.

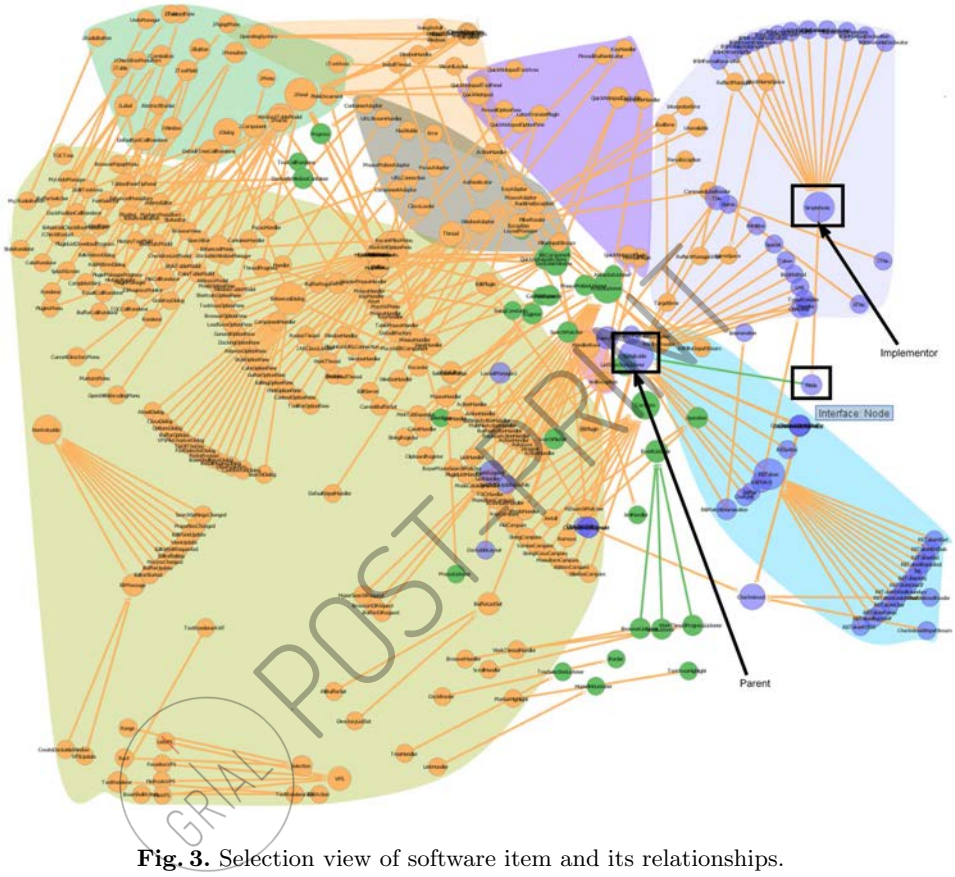

Fig. 3. Selection view of software item and its relationships.

5 Conclusions

The visual representation proposed on this paper allows to answer several questions at a glance without much effort. It supports the visualization of the project structure, dependency information and highlights project features.

This visualization is effective for the discovery of relationships among software items and determining the impact of one software item on other software

items due to inheritance level and feature propagation when changes take place. The representation is powerful because of its simplicity, intuitive design and support to make evident what usually is not easy to identify on large software projects. It is part of a larger visualization proposal and contributes to the comparison of structure changes on software projects when two or more revisions are selected.

The main contributions of the proposal are the uncovering and representation of hierarchy and implementation relationships between Java software items, within the software project and external software items that belongs to the Java API or libraries used by the project under consideration.

References

1. Etzioni, O.: The World-Wide Web: quagmire or gold mine? Communications of the ACM **39**(11) (1996) 65–68
2. Estublier, J.: Software Configuration Management: A Roadmap. The Future of Software Engineering (2000) ISBN 1-58113-253-0.
3. Collins-Sussman, B., Fitzpatrick, B., Pilato, M.: Version Control with Subversion. Sebastopol, CA USA: O'Reilly Media, Inc. (2004) ISBN: 0-596-00448-6.
4. Kagdi, H., Collard, M., Maletic, J.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance and Evolution: Research and Practice **19**(2) (March/April 2007)
5. Xie, X., Poshyvanyk, D., Marcus, A.: Visualization of CVS Repository Information. In: WCRE '06: Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006), Benevento, Italy, IEEE Computer Society (October 2006) 231–242
6. Therón, R., González, A., García, F.J., Santos, P.: The Use of Information Visualization to Support Software Configuration Management. In Baranauskas, C., Palanque, P., Abascal, J., Barbosa, S., eds.: Human-Computer Interaction - INTERACT 2007. Proceedings of the 11th IFIP TC 13 International Conference. Volume 4663 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (September 2007) 317–331 ISBN: 978-3-540-74799-4.
7. Voinea, L., Telea, A.: CVSgrab: Mining the History of Large Software Projects. In Santos, B.S., Ertl, T., Joy, K.I., eds.: EuroVis06: Joint Eurographics - IEEE VGTC Symposium on Visualization, Lisbon, Portugal, Eurographics Association (May 2006) 187–194
8. Weissgerber, P., Diehl, S., Zimmermann, T., Zeller, A.: Mining Version Histories to Guide Software Changes. IEEE Transactions on Software Engineering **31**(6) (June 2005) 429–445 Student Member-Thomas Zimmermann and Member-Andreas Zeller, ISSN: 0098-5589.
9. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley (May 1999)
10. Dürsteler, J.: Visualización del Contenido de la Web. Revista Digital de InfoVis.net (175) (2005)
11. Therón, R.: Hierarchical-temporal Data Visualization using a Tree-Ring Metaphor. In Butz, A., Fisher, B., Krüger, A., Olivier, P., eds.: Smart Graphics. Proceedings of the 6th International Symposium, SG 2006. Volume 4073 of Lecture Notes in Computer Science., Springer Verlag (July 2006) 70–81 ISBN: 978-3-540-36293-7.

12. Marcos, M.: Visual Elements in Search and Information Retrieval Systems. Hipertext.net (3) (2005)
13. Kobayashi, M., Takeda, K.: Information Retrieval on the Web. ACM Computing Surveys **32**(2) (2000) 144–173
14. Henzinger, M.: Information Retrieval on the Web. In: 39th Annual Symposium on Foundations of Computer Science (FOCS98), Palo Alto, CA (1998)
15. Motro, A.: INFS-623: Classical and Web Information Retrieval. Information Retrieval from the Web. Technical report, George Mason University, Computer Science Department (2007)
16. Paltoglou, G., Salampasis, M., Satratzemi, M.: A results merging algorithm for distributed information retrieval environments that combines regression methodologies with a selective download phase. Information Processing & Management **44**(4) (July 2008) 1580–1599
17. Pokorný, J.: Web Searching and Information Retrieval. Computing in Science and Engineering **6**(4) (2004) 43–48
18. Glover, E., Lawrence, S., Birmingham, W., Giles, C.: Architecture of a Metasearch Engine that supports user information needs. In: CIKM '99: Proceedings of the Eighth International Conference on Information and Knowledge Management, New York, NY, USA, ACM (1999) 210–216
19. Vesperman, J.: Essential CVS. Sebastopol, CA USA: O'Reilly Media, Inc. (2006) ISBN: 0-596-52703-9.
20. IEEE: IEEE Standard for Software Configuration Management Plans. IEEE Std 828-2005 (Revision of IEEE Std 828-1998) (2005) 1–19
21. Voinea, L., Telea, A.: An open framework for cvs repository querying, analysis and visualization. In: MSR '06: Proceedings of the 2006 international workshop on Mining software repositories, New York, NY, USA, ACM Press (2006) 33–39
22. González-Barahona, J.M., Robles, G., Herraiz, I.: Challenges in Software Evolution: the Libre Software Perspective. In: International ERCIM-ESF Workshop on Challenges in Software Evolution (ChaSE). (2005)
23. German, D.M., Cubranić, D., Storey, M.A.D.: A framework for describing and understanding mining tools in software development. In: MSR '05: Proceedings of the 2005 International Workshop on Mining Software Repositories, New York, NY, USA, ACM (2005) 1–5
24. Leung, Y.K., Apperley, M.D.: A review and taxonomy of distortion-oriented presentation techniques. ACM Trans. Comput.-Hum. Interact. **1**(2) (1994) 126–160
25. Tufte, E.: Envisioning information. Graphics Press, Cheshire, CT, USA (1990)
26. Tufte, E.R.: Visual explanations: images and quantities, evidence and narrative. Graphics Press, Cheshire, CT, USA (1997)
27. Voinea, L., Telea, A.: Mining software repositories with CVSgrab. In: MSR '06: Proceedings of the 2006 international workshop on Mining software repositories, New York, NY, USA, ACM Press (2006) 167–168
28. Sangal, N., Jordan, E., Sinha, V., Jackson, D.: Using dependency models to manage complex software architecture. SIGPLAN Not. **40**(10) (2005) 167–176
29. Pei-Breivold, H., Crnkovic, I., Land, R., Larsson, S.: Using Dependency Model to Support Software Architecture Evolution. In: 4th International ERCIM Workshop on Software Evolution and Evolvability (Evol08), L'Aquila, Italy, IEEE (September 2008)
30. Rysselberghe, F.V., Demeyer, S.: Studying Versioning Information to Understand Inheritance Hierarchy Changes. In: MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories, Minneapolis, MN, USA, IEEE Computer Society (May 2007) ISBN:0-7695-2950-X.