Einreichung für: Workshop Industrial Applications of Artificial Intelligence 2017
Geplant als Veröffentlichung innerhalb der Lecture Notes in Informatics (LNI)

Mining Java Packages for Developer Profiles: An Exploratory Study

Jasmin Ramadani[1] and Stefan Wagner[2]

Abstract: Not all developers have the same degree of knowledge of all parts of a software system. For allocating new task expertise, it would be interesting to have different developer profiles explicit. The state of the practice is to find out manually who might be most experienced in a certain area. A clear understanding how to automate this analysis is missing. Our goal is to explore to what degree the analysis of couplings of packages can be useful for this automation. Our analysis approach uses the idea that packages reflect the organization of the source code into basic functionalities. We use data mining on the version history to identify the sets of the packages that were most frequently changed together in different occasions. We present a case study where we analyze three open-source software systems to define developer expertise profiles based on the aggregation of the packages. Our results identify different developer profiles. They can be especially useful in analyzing projects with a larger number of developers saving time and effort by limiting the data sets to be investigated to find relevant software changes.

Keywords: Version history;Packages;Developers expertise

1 Introduction

Software systems are developed and changed by a number of developers over time. The contribution of developers to a project consists of the combination of their actions performed during software development [GKS08]. This kind of expertise is called *implementation expertise* [SZ08]. Yet, software developers involvement in the source code becomes diverse and awkward to follow. As the development team grows, it becomes more complicated to allocate the developers according to their expertise considering their contribution to the project. One approach defines that the developers who changed a file are considered to have the expertise for that file [SZ08]. This information can be found by mining changes in the software repositories.

In Java projects, packages are important to group source code based on its functionality. By scanning the changes in the versioning system, we can extract the packages of files being changed. For the reason that the package structure is more stable and does not change often, we use the packages instead of classes or methods to define developer expertise profiles.

An approach how to find file changes that happened together frequently using data mining has been described in [KYM06; RW16b; Yi04]. The changes are grouped based on the developer that performed the commits. These changes are called *coupled file changes* and

[1] University of Stuttgart, jasmin.ramadani@informatik.uni-stuttgart.de
[2] University of Stuttgart, stefan.wagner@informatik.uni-stuttgart.de

can be offered to the developers as recommendations when solving some maintenance task. We use this technique to extract the packages which changed most frequently together to define developer profiles. The profiles can be used to identify who worked on similar issues on the project.

1.1 Problem Statement

Developers working on maintenance tasks could use the information who worked on which part of the source code to assign work or ask for help. Having many developers on a project, it is difficult to identify the experts working on topics related to their tasks.

1.2 Research Objective

The aim of our study is to identify the developers' expertise profiles based on the projects' package structures. We use the package organization of the source code because it reflects the grouping of the system functionalities and defines the fundamental features or layers of the system. Using data mining, we investigate the software repositories to extract the sets of packages that were most frequently changed together by a given developer during software development. The fact that these sets of package changes are repeating in different occasions give us better expertise description than simply listing how often the packages have been changed by the developer. Based on the functionalities behind these sets, we differentiate the developer expertise.

1.3 Contribution

We present an exploratory case study where we define developer profiles of expertise based on the aggregated information about the system packages that were most frequently changed together. By differentiating developer profiles, the effort for the analysis drops because the users involved in maintenance do not have to examine the data from all developers on the project. They can choose the data from those developers who implemented changes in the system relevant to their tasks.

2 Background

2.1 Java Packages

Packages in Java represent namespaces to organize set of classes and interfaces which prevent conflicts in the source code. They can be related according to some specific functionality. For example, the *java.io* package groups the classes dealing with input and output. There are two fundamental approaches for creating packages in Java. Package by feature reflects a set of features of the software. It organizes the classes related to a single feature. Package by layer reflects various application levels instead of features.

2.2 Data Mining

Hidden dependencies between source code files are also known as logical dependencies or couplings [GHJ98]. Coupled file changes describe a situation where someone changes a file and afterwards also changes another file. We use the same principle to investigate the version history for frequent change sets on a package instead of file level. One of the most popular data mining techniques is the discovery of frequent item sets.

We use a heuristic where the commits containing the package names are grouped based on the developers who committed the changes. The number of developers identifies the number of groups of commits for which the mining process will be performed. The user-defined support threshold identifies the frequency of the package couplings.

Tab. 1: Developer Profiles

	Package	Profile
Dev.1	astpa/src/astpa/controlstructure/controller/ astpa/src/astpa/controlstructure/create astpa/src/astpa/controlstructure/command/ astpa/src/astpa/controlstructure/controller/editParts/ astpa/src/astpa/controlstructure/policy/	control structure
Dev.2	astpa/src/astpa/ui/interfaces/ astpa/src/astpa/ui/common/grid astpa/src/astpa/ui/sds/	user interface

2.3 Developer Profiles

Developers working on a project can be differentiated based on their contribution to the functionalities of the software. There are developers that commit a significant number of commits to the system whereby others only contribute [ADG08]. Existing approaches investigate developer expertise based on the authorship of these changes [Fr10]. In our study, we define profiles of developers expertise on the level of packages. Using developers' profiles we capture the characteristics of their changes according to their contribution in the source code. Schuler et al. [SZ08] defined a process for aggregating developer expertise profiles. Similarly, we use the changed packages from the version history to aggregate the expertise profile of the developer. The profile contains all coupled packages changed by the developer. We rank the coupled packages to find the most frequent package couplings to define the profile.

Users working on some maintenance task can use the profiles to decide which developer's data given the implementation expertise matches to their work the best. For example, lets suppose that a developer has some maintenance task related to some change in the *control structure* of the application. He or she looks up at the developers profiles to find the developer whose most frequent package couplings are relevant for his feature. In Table 1 we have two developers, their most frequently changed packages and the defined profiles. We see that the most frequently coupled packages by the first developer cover features related to the *control structure*. We aggregate the features covered in these packages in a developer profile: *control structure*. The second developer profile describes package features related to the *user interface* and are not relevant for the task mentioned above.

3 Related Work

Various studies define the developer expertise based on the changes in the code. McDonald [Mc01] presents a recommendation system using the change history information based on the Line 10 rule. Here, the developer whose name shows up on line 10 of the change log is the last person who worked on the file. A developer is considered to be an expert if he or she is the last person having the file in memory [RR13]. Gitba et al. [Gi05] measure the code ownership based on who is the last committer. We consider all the developers working on a particular part of the code, not only the last one. A similar approach is defined by Mockus and Herbsleb [MH02] where the number of times the file is changed by a developer is measured for the expertise. Nakakoji et al. [Na02] define an onion like structure to describe the contributing developers to a project. Our approach differs because we do not investigate methods or files to define developer profiles but Java packages. We investigate frequently changed couplings of packages and not only the number of changes.

Manto and Murphy propose developer expertise [MM07] where the expert is defined using data from versioning system and explore how often a developer changed a particular file. The study by Schuler et al. [SZ08] recommends experts by mining of version archives based on the methods they have changed. Another approach, presented by Anvik and Murphy [AM07] defines a three-way approach using the source repository check-in logs, bug reports and the modules containing changes determined on file or package level. Alonso et al. [ADG08] uses CVS to identify and visualize developer expertise. Similarly, we use the version history of the project, in our case it is the Git repository. However, we investigate not on a method but on a package level. Joblin et al. presents an empirical study about classification of developers in to core and peripheral roles [Jo16].

There are several studies where the package information in the project has been investigated. Robles et al. [Ro06] studied the characteristics of software including the packages, lines of codes or the programming language. In our study, we do not describe the content of the classes in the packages, we identify the way the source code is divided in packages based on the features or layers. The project analysis studies performed in [Ha08; SD07] involve dependency analysis between the packages. We investigate logical couplings between the packages and not architectural dependencies.

Most of the studies dealing with identifying coupled changes from software repositories use some kind of data mining for this purpose [KYM06; RW16a; RW16b; Yi04; Zi04]. In [KYM06; Yi04] the authors investigate the couplings based on the file level. In [FGP05; Zi04] the researchers identify couplings between parts of files like classes, methods or modules. We investigate higher level couplings based on the project packages.

4 Case Study Design

4.1 Research Questions

RQ1: Which package couplings are most frequent per developer? We examine which packages are most frequently changed together by particular developers. We use this

information to investigate the functionalities the developers were mostly involved during the software development.

RQ2: What kind of developer profiles can we define based on the packages? Based on the changed packages and their functionalities, we aggregate them to define their profiles related to their expertise on the system software. Using this information, developers can explicitly identify the software changes related to their task.

4.2 Case Selection

For our study, we use three open source projects: ASTPA, an Eclipse based Java project, RIOT, Java and Android based software and VITA, Java based text analysis software. They were all developed at the University of Stuttgart and were found on the local Gitlab. The projects have been selected based on their structure having a number of packages and developers and their availability for the study.

4.3 Data Collection Procedure

We extract the Git log from the project repositories and format the output to separate the commits according to the developer who has done the changes. We enlist the commits with all changed files represented by the file names containing the relative file paths including the packages and sub-packages of the project. To prepare the data for analysis, we remove empty commits or commits having only a single entry. We group the commits based on the developer heuristics. We create data sets of commits for every developer who contributed to the repository.

4.4 Analysis Procedure

- *Extracting the packages from the commits:* We extract all names of the files changed in a commit. They include the relative file path on the system which includes the names of the package and sub-packages. We scan the file paths from the back and remove the filenames from the path. The resulting string identifies the name of the package or sub package.
- *Mining packages:* We perform a frequent item sets analysis on the packages to extract the most frequent couplings from the the repository. Due to the high number of file couplings, we use a relatively high user-defined support level for the frequent item sets analysis. This way we include only the coupled packages which happened frequently and not by chance.
- *Define developer profiles:* We rank all the coupled packages for a developer starting with the most frequently changed package to identify the most frequent ones. After the ranking of the packages, we join the group of most frequent packages to the developer. This will identify the expertise of the developer marking the functionalities

he or she was most involved into. We look up the features that are involved in the files behind this packages. We aggregate developers expertise profile based on the most frequently changed packages.

5 Results and Discussion[3]

We have extracted the most frequent package couplings for the developers in all three projects. Depending on the outcomes of the mining algorithm, we set an average support level of 80% for the first project, 60% for the second and 40% for the third project. These values define the strength of the couplings.

Our results show that the number of different coupled packages vary from developer to developer and from project to project. These values are mainly influenced by the number of functionalities the developers have been involved into.

Tab. 2: Frequent packages and expertise profiles

	ASTPA	
	packages	profiles
Dev1	astpa.src.astpa.controlstructure.controller.editParts	control structure
Dev2	astpa.src.astpa.controlstructure astpa.src.astpa.controlstructure.controller.editParts astpa.src.astpa.controlstructure.figure	control structure
Dev3	astpa.src.astpa.controlstructure.controller.editParts astpa.src.astpa.model.interfaces astpa.src.astpa.ui.common.grid	control structure+ user interface+model
Dev4	astpa.src.astpa.model.interfaces	user interface+model
Dev5	astpa.astpa.intro.graphics.icons	graphics
Dev6	astpa.src.astpa.ui.sds, astpa.src.astpa.ui.acchaz	user interface
Dev7	astpa.src.astpa.ui.common.grid	user interface
Dev8	astpa.icons.buttons.systemdescription	graphics
	RIOT	
Dev.1	android.riot.res.layout, android.riot.res.drawable-xhdpi android.riot.res.drawable-mdpi, android.riot.res.drawable-hdpi, android.riot	android layout
Dev.2	android.res.layout android.res.layout android.src.main.java.de.uni_stuttgart.riot.android.management	android layout
Dev.3	commons.src.main.java.de.uni_stuttgart.riot.server.commons.db	database
Dev.4	android.riot, android.riot.settings, android.riot.res.drawable-hdpi android.riot.res.drawable-mdpi, android.riot.res.drawable-xhdpi android.riot.res.drawable-xxhdpi, android.riot.res.layout android.riot.res.menu, android.riot.res.values-de	android layout
Dev.5	android.src.main.java.de.uni_stuttgart.riot.android.account	android account
Dev.6	android.src.main.java.de.uni_stuttgart.riot.android.idea.libraries .idea.libraries	android libraries
Dev.7	usermanagement.src.main.java.de.uni_stuttgart.riot. usermanagement.data.sqlQueryDao.impl	database
Dev.8	commons.src.main.java.de.uni_stuttgart.riot.thing	riot things
Dev.9	webapp	webapps
Dev.10	commons.src.main.resources.languages, webapp	webapps
	VITA	
Dev.1	src.main.resources.gate_home.plugins.annie.resources.gazetteer	ANN plugins
Dev.2	src.main.java.de.unistuttgart.vis.vita.services	services
Dev.3	src.main.front-end.app.partials	frontend
Dev.4	src.main.java.de.unistuttgart.vis.vita.analysis	analysis
Dev.5	src.main.java.de.unistuttgart.vis.vita.importer.epub	importer
Dev.6	src.main.java.de.unistuttgart.vis.vita.importer.epub	importer
Dev.7	src.main.front-end.app.partials	frontend

[3] The complete list of all couplings and profiles are available at: http://dx.doi.org/10.5281/zenodo.51302.

5.1 Most frequent package couplings per developer (RQ1)

For every developer, we have extracted a list of package couplings covering various software features. For example, in Table 2, we see that the first developer on the ASTPA project changed mostly files in the *controller.editParts* sub-package, whereby the sixth and the seventh developer changed mostly the *ui* related packages.

The first, the second and the fourth developer who worked on the RIOT project changed very similar packages related to the layout whereby the third and the seventh developer worked on the database packages.

The second developer who worked on the VITA project worked on the services whereby the third and the seventh developer worked on packages related to the front-end. The results show that some of them changed functionalities in files that belong to the same package, whereby others worked on different packages. Some of the developers share the packages meaning that two or more developers worked on the same packages. In other cases the developers split their work and contributed into totally different packages.

5.2 Developer profiles (RQ2)

Based on most frequent package couplings we have aggregated various developer profiles for all three projects as presented in Table 2. For the first project we identify profiles of developers related to the changes on the control structure, user interface and graphics. For the second project we define developer profiles covering expertise on the android layout, database, account, libraries, android functionalities and web apps. For the developers on the third projects we define profiles including involvement in the plugins, services, front end, analysis and import features.

The difference in the profiles is directly influenced by the areas of the system where the developers have worked on. Also the project organization and the number of packages affect the precision of the profiles.

5.3 Discussion

Using the most frequently changed packages, various functionality topics in the source code structure have appeared. For some developers we have limited set of changed system functionalities which gives clear information what they were mainly working on. This way, the profiles related to a small number of functionalities and show a more precise developer expertise. This is useful to define a precise expertise profile and clearly identifies changes related to a particular maintenance task for developers working on a concrete part of the system.

Other package changes show developers working on various topics in the system. They show more general expertise working on different areas of the system. Their profiles do not

clearly represent a set of changes related to a specific functionality. They are based on a set of changes covering broader system contribution. However, they can be still valuable for developers working on various parts of the software or on new functionalities.

6 Threats to Validity

A threat to construct validity could be that the developers by creating the packages influence the organization of the code. The names of the packages could lead to a group of classes which are not related or similar. We look up in the classes and other files in the packages to inspect manually if they are related to the package name.

As an internal threat we can mention that the relatively high support level for the data mining algorithms provides relative small number of package couplings. However, this ensures that these couplings happened frequently and not by chance.

The generalization of our approach represents an external threat because it is limited and we investigated a small number of Java projects. However, it is possible to apply our approach on any project having a clear package or namespace structure and having several developers working on it.

7 Conclusion

We conclude that we can successfully define developer profiles based on the packages that most frequently changed together. There are developers working on similar system functionalities which can lead to more precise developer profiles. This can limit the number of data sets for data mining, which decreases the effort for the analysis and can be very helpful in projects with a large number of developers. For the developers working on various or totally different parts of the code, the profiles show more general coverage of system expertise.

The resulting profiles differentiate a number of functionalities, which can help the users working on their maintenance tasks to choose the most relevant implementation. The next steps will be to formalize the automation of the expertise analysis process of the profile description according to the package structure of the system.

References

[ADG08] Alonso, O.; Devanbu, P. T.; Gertz, M.: Expertise Identification and Visualization from CVS. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories. MSR '08, pp. 125–128, 2008.

[AM07] Anvik, J.; Murphy, G. C.: Determining Implementation Expertise from Bug Reports. In: Proceedings of the Fourth International Workshop on Mining Software Repositories. MSR '07, pp. 2–, 2007.

[FGP05] Fluri, B.; Gall, H.; Pinzger, M.: Fine-Grained Analysis of Change Couplings. In: SCAM. Pp. 66–74, 2005.

[Fr10] Fritz, T.; Ou, J.; Murphy, G. C.; Murphy-Hill, E.: A Degree-of-knowledge Model to Capture Source Code Familiarity. In: Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1. ICSE '10, pp. 385–394, 2010.

[GHJ98] Gall, H.; Hajek, K.; Jazayeri, M.: Detection of Logical Coupling Based on Product Release History. In: Proceedings of the International Conference on Software Maintenance. ICSM '98, pp. 190–, 1998.

[Gi05] Girba, T.; Kuhn, A.; Seeberger, M.; Ducasse, S.: How Developers Drive Software Evolution. In: Proceedings of the Eighth International Workshop on Principles of Software Evolution. IWPSE '05, pp. 113–122, 2005.

[GKS08] Gousios, G.; Kalliamvakou, E.; Spinellis, D.: Measuring Developer Contribution from Software Repository Data. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories. MSR '08, 2008.

[Ha08] Hautus, E.: Improving Java Software Through Package Structure Analysis, 2008, URL: http://ehautus.home.xs4all.nl/papers/PASTA.pdf/.

[Jo16] Joblin, M.: Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics, 2016, URL: http://arxiv.org/abs/1604.00830.

[KYM06] Kagdi, H.; Yusuf, S.; Maletic, J. I.: Mining Sequences of Changed-files from Version Histories. In: Proceedings of the 2006 International Workshop on Mining Software Repositories. MSR '06, pp. 47–53, 2006.

[Mc01] McDonald, D. W.: Evaluating Expertise Recommendations. In: Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work. GROUP '01, pp. 214–223, 2001.

[MH02] Mockus, A.; Herbsleb, J. D.: Expertise browser: a quantitative approach to identifying expertise. In: ICSE '02: Proceedings of the 24th International Conference on Software Engineering. Pp. 503–512, 2002.

[MM07] Minto, S.; Murphy, G. C.: Recommending Emergent Teams. In: Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007). Pp. 5–5, 2007.

[Na02] Nakakoji, K.; Yamamoto, Y.; Nishinaka, Y.; Kishida, K.; Ye, Y.: Evolution Patterns of Open-source Software Systems and Communities. In: Proceedings of the International Workshop on Principles of Software Evolution. IWPSE '02, pp. 76–85, 2002.

[Ro06] Robles, G.; Gonzalez-Barahona, J. M.; Michlmayr, M.; Amor, J. J.: Mining Large Software Compilations over Time: Another Perspective of Software Evolution. In: Proceedings of the International Workshop on Mining Software Repositories (MSR 2006). Pp. 3–9, 2006.

[RR13] Robbes, R.; Rothlisberger, D.: Using developer interaction data to compare expertise metrics. In. Pp. 297–300, 2013.

[RW16a] Ramadani, J.; Wagner, S.: Are Suggestions of Coupled File Changes Interesting? In: Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering. Pp. 15–26, 2016.

[RW16b] Ramadani, J.; Wagner, S.: Which Change Sets in Git Repositories Are Related? In: International Conference on Software Quality, Reliability and Security (QRS). 2016.

[SD07] S.Ducasse; D.Pollet; M.Suen; andI. Alloui, H.: Package Surface Blueprints: Visually Supporting the Understanding of Package Relationships. In: 2007 IEEE International Conference on Software Maintenance. Pp. 94–103, 2007.

[SZ08] Schuler, D.; Zimmermann, T.: Mining Usage Expertise from Version Archives. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories. MSR '08, pp. 121–124, 2008.

[Yi04] Ying, A. T. T.; Murphy, G. C.; Ng, R. T.; Chu-Carroll, M.: Predicting Source Code Changes by Mining Change History. IEEE Transactions on Software Engineering 30/9, pp. 574–586, 2004.

[Zi04] Zimmermann, T.; Weisgerber, P.; Diehl, S.; Zeller, A.: Mining Version Histories to Guide Software Changes. In: Proceedings of the 26th International Conference on Software Engineering. ICSE '04, pp. 563–572, 2004.