Developer Experience with the Django Web Framework
(Extended Abstract)

Frederik Rüther Hakan Aksu Ralf Lämmel
Software Languages Team, University of Koblenz-Landau

Abstract

Web frameworks involve many aspects, e.g., forms, model, testing, and migration. Developers differ in terms of their per-aspect experience. We describe a methodology for the identification of relevant aspects of a web app framework, measurement of experience atoms per developer and per aspect based on the commit history of actual projects, and the compilation of developer profiles for summarizing the relevance of different aspects and the developers' contribution to the project. Measurement relies on a rule-based language. Our case study is concerned with the Python-based Django web app framework and the open source Django-Oscar project from which experience atoms were extracted.

1 Motivation

A typical scrum team consists of developers, a scrum master, a product owner, and yet others. One of the typical tasks of a product owner is to organize the product backlog which may also include bugs. A task of the product owner is to assign bugs to persons. Finding suitable persons may be challenging for large teams with significant fluctuation. Our approach may recommend a developer for the assignment based on the analyzed experience. We assume that necessary experience can also be extracted from bug reports and associated files by essentially the same method. We apply our approach in a case study to the Django app framework and to the Django-Oscar project on Github from which experience atoms are extracted.

2 Experience atoms

The key idea is to rank developers in terms of aspects based on their past commit activities. Experience is measured in experience atoms per aspect. We rely on the following definition [2]: "Experience atoms are elementary units of experience. Experience, we assume, is the direct result of a persons activity with respect to a work product, enhancing it or fixing a problem. The smallest meaningful unit of such changes is an experience atom."

The process of finding aspects is based on two steps. First, a suitable documentation such as a tutorial

Figure 1: Classification of aspects of experience in Django projects

(e.g., the Django tutorial[1] in the case study) is used as input; a list of mentioned classes, directories, and files is aggregated, subject to a threshold for inclusion. Second, all the imports from the project under investigation are collected. The list of imports is used to identify widely used aspects that are not part of the documentation from the first step. Both lists are combined and names are assigned to the items; see Fig. 1 for the result.

3 Rule-based detection

The analysis of a project is based on rules. A rule can be described as a function that maps source code changes along commits to identified experience atoms. Such functions are composed from filters operating at different source-code levels and operations that combine Boolean values and numbers of experience atoms.

There exist four main filters:

FilenameFilter a condition on the filename of changed file.

DirectoryFilter a condition on the directory of a changed file.

[1] https://docs.djangoproject.com/en/1.9/intro/tutorial01/

TextFilter a condition on the text-level content of a changed file.

TreeFilter a condition on the parse tree-level content of a changed file. For instance, one may search for specific methods or classes for the sake of detecting experience atoms for aspects.

Those filters can be combined with the help of operations 'and', 'or', and 'implies'. Here is an example for a rule that uses two filters:

Other/KnowledgeManagement:
DirectoryFilter("docs") **or**
FilenameFilter("ReadMe.md")

This rule identifies the aspects *KnowledgeManagement* if a commit took place in the *docs* folder or a changed file has the name *ReadMe.md*. The filters are combined with an *or* operation since both operations indicate that the person has performed a task related to knowledge management. The number of changed lines are to be counted as experience atoms.

A recurring detection scenario is that an aspect is associated with inheritance from a certain framework class. To this end, we leverage a **SuperClassFilter** which is derived from the basic filter *TreeFilter* and checks for a specific superclass. For instance:

Model/Data:
TextFilter("import django.db.Model") **implies**
SuperClassFilter("Model")

In this rule, we also check that a specific namespace was imported to avoid confusion of the *Model* class with one from another library.

Aspect	Category	Count	Unit	%
VCS	Administration	134	file	88
HTML	Forms	1205	class	55
Validation	Other	10	function	11
Model	Forms	1827	class	54
BuildM.	Administration	300	file	66
DjangoConfig.	Other	3005	file	88
Fixture	Database	4923	file	92
Templates	Other	27140	count	42
Django	Test	12703	class	72
AdminInterface	Model	770	class	82
Migration	Database	58	class	14
Generell	Test	16091	file	67
Data	Model	20683	class	61
KnowledgeM.	Other	4962	file	57

Figure 2: Experience of David Winterbottom in the Django-Oscar project. The last column shows the percentage of the experience for an aspect in the project. The units 'file', 'class', and 'function' refer to the number of changed lines in the corresponding scope; the unit 'count' refers to the number of matches in the changed lines.

4 Developer profile

Fig. 2 shows the experience of the developer with the most commits in the Django project of our case study. While he collected usually the most experience atoms, there are three aspects where he does not own over 50 %, namely *Templates*, *Migration*, and *Validation*. This observation gives insight into the collaboration and the division of responsibilities in the project.

5 Related work

Our work on rule-based detection of experience aspects ('skills') is inspired, for example, by previous work of Teyton et al. [4]. In this work, skills related to Java development, e.g., development of JUnit tests, are analyzed also with the help of a rule-based language and also based on analyzing the commit history. No general classification of experience aspects is developed though; this work was targeted at industrial case studies that were using the Java platform.

Matter et al. [1] assign bugs to developers by the similarity of the vocabulary in their produced source code and the bug description. Nguyen et al. [3] track bug reports through their lifecycle. Based on such historical data, a recommendation of a developer for a future assignment is provided. Our approach introduces explicitly a high level of abstraction in terms of an up-front classification of experience, subject to rule-based detection.

6 Conclusion

Let us return to the motivating scenario on bug assignment. Given relatively dominant developers such as David Winterbottom in our case study, it may be an obvious choice to assign 'all' bugs to these dominators. Obviously, such a naive approach does not scale. In the interest of a division of work, the developer profiles are to be used to find the 'next best' assignment, i.e., someone who is sufficiently experienced and available overall. Further, the developer profiles also reveal aspects (for dominant developers or otherwise) for which other developers would be more 'qualified' (more experienced) anyhow.

In future work, we plan to validate our approach in terms of the feasibility and accuracy of assigning developers to bugs. For instance, we would like to compare suggestions based on our approach with some available ground truth based on project history.

References

[1] Dominique Matter, Adrian Kuhn, and Oscar Nierstrasz. Assigning Bug Reports Using a Vocabulary-based Expertise Model of Developers. In *Proc. MSR 2009*. IEEE Computer Society, 2009.

[2] Audris Mockus and James D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proc. ICSE 2002*, pages 503–512. ACM, 2002.

[3] Tung Thanh Nguyen, Tien N. Nguyen, Evelyn Duesterwald, Tim Klinger, and Peter Santhanam. Inferring developer expertise through defect analysis. In *Proc. ICSE 2012*, pages 1297–1300. IEEE, 2012.

[4] Cédric Teyton, Marc Palyart, Jean-Rémy Falleri, Floréal Morandat, and Xavier Blanc. Automatic extraction of developer expertise. In *Proc. EASE 2014*, pages 8:1–8:10. ACM, 2014.