Feature Defect Prediction

Stefan Strüder
University of Koblenz-Landau, Germany

Daniel Strüber
Radboud University Nijmegen, Netherlands

Thorsten Berger
Chalmers | University of Gothenburg, Sweden

Mukelabai Mukelabai
Chalmers | University of Gothenburg, Sweden

ABSTRACT

Software errors are a major nuisance in software development and can lead not only to damage of reputation but also to considerable financial losses for companies. For this reason, numerous techniques for detecting and predicting errors have been developed over the past decade, which are largely based on machine learning methods. The usual approach of these techniques is to predict errors at file level. For some years now, however, the popularity of feature-based software development has been increasing - a paradigm that relies on function increments of a software system (features) and thus ensures a wide variability of the software product. A common implementation technique for features is based on annotations with preprocessor instructions, such as `#IFDEF` and `#IFNDEF`, whose code is spread over several files of the software's source code ("code scattering"). A bug in such a feature code can have far-reaching consequences for the functionality of the entire software. If a part of the feature code contains errors, the entire function of the feature becomes faulty and may lead to the failure of the entire functionality of the software (features are "cross-cutting"). This problem is the subject of this thesis. A prediction technique for faulty features is developed, which is based on methods of machine learning. The evaluation of eight classifiers, each based on an individual classification algorithm, shows that the feature-based data set created for this thesis allows an accuracy of up to 92% for the prediction of faulty or error-free features. It is also shown how the feature orientation aspect was incorporated into the creation of the dataset and what results were achieved compared to the traditional file-based methodology.

ACM Reference Format:
Stefan Strüder, Daniel Strüber, Thorsten Berger, and Mukelabai Mukelabai. 2020. Feature Defect Prediction. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

2 BACKGROUND / RELATED WORK

3 METHODOLOGY

3.1 Creation of Dataset

The data set forms the basis for the training of the machine learning classifiers and is created specifically for this work based on commits from 13 feature-based software projects. The software projects are selected on the basis of previous use in scientific literature [1–4]. The software projects used for this thesis are listed in Table 1 together with their purpose and data sources.

Conference'17, July 2017, Washington, DC, USA
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00

Table 1: Used software projects

Project	Purpose	Data source
Blender	3D modelling tool	GitHub mirror
Busybox	UNIX toolkit	Git repository
Emacs	text editor	GitHub mirror
GIMP	graphics editor	GitLab repository
Gnumeric	spreadsheet	GitLab repository
gnuplot	plotting tool	GitHub mirror
Irssi	IRC client	GitHub repository
libxml2	XML parser	GitLab repository
lighttpd	web server	Git repository
MPSolve	polynom solver	GitHub repository
Parrot	virtual machine	GitHub repository
Vim	text editor	GitHub repository
xfig	graphics editor	Sourceforge repository

To get the commit data of the software projects the Python library PyDriller[1] was used [5]. This allows easy data extraction from Git repositories to obtain commits, commit messages, developers, diffs, and more (called "metadata" in the following). The URLs to the Git repositories of the software projects were used as input for the specially created Python scripts for receiving the commit metadata. Furthermore, the metadata was divided into commits per release. This was made possible by specifying release tags in the PyDriller code, based on the tag structure of Git repositories. For each modified file within a commit and a release, the following metadata was retrieved using PyDriller:

- commit hash (unique identifier of a commit)
- commit author
- commit message
- filename
- lines of code
- cyclomatic complexity
- number of added lines
- number of removed lines
- diff (changeset)

The data obtained in this way was stored in a MySQL database after retrieval. For each software project, a separate table was created in which, in addition to the metadata above, the name of the software project and the release numbers associated with the commits were stored. Each modified file of a commit receives one row of the database tables. The further construction of the data set is divided into several phases of data processing and optimization.

[1] https://github.com/ishepard/pydriller

The first phase consists of extracting the features involved in a modified file. This was done by using a Python script to identify the preprocessor statements `#IFDEF` and `#IFNDEF` in the diffs of the modified files, and then saving the string following the directives as a feature until the end of the line of code. The identification was done using regular expressions. The features identified per file are stored in an additional column in the respective MySQL tables. In case a feature is identified after the `#IFNDEF` directive, the feature is stored with a preceding "not". It will be saved as a separate feature, along with its non-negated form. Combinations of features are stored in their identified form. If no feature could be identified, `"none"` is saved accordingly.

3.2 Selection of Metrics

3.3 Selection of Classifiers

4 EVALUATION

5 DISCUSSION

6 CONCLUSION

REFERENCES

[1] Claus Hunsen, Bo Zhang, Janet Siegmund, Christian Kästner, Olaf Leßenich, Martin Becker, and Sven Apel. 2015. Preprocessor-based variability in open-source and industrial software systems: An empirical study. *Empirical Software Engineering* 21, 2 (apr 2015), 449–482. https://doi.org/10.1007/s10664-015-9360-1

[2] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. 2010. An analysis of the variability in forty preprocessor-based software product lines. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*. ACM Press. https://doi.org/10.1145/1806799.1806819

[3] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. 2016. Towards predicting feature defects in software product lines. In *Proceedings of the 7th International Workshop on Feature-Oriented Software Development - FOSD 2016*. ACM Press. https://doi.org/10.1145/3001867.3001874

[4] Rodrigo Queiroz, Leonardo Passos, Marco Tulio Valente, Claus Hunsen, Sven Apel, and Krzysztof Czarnecki. 2015. The shape of feature code: an analysis of twenty C-preprocessor-based systems. *Software & Systems Modeling* 16, 1 (jul 2015), 77–96. https://doi.org/10.1007/s10270-015-0483-z

[5] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. PyDriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*. ACM Press. https://doi.org/10.1145/3236024.3264598

Table 2: Overview of used metrics

	Metric	Description	Source
Process metrics	Number of commits (COMM)	Number of commits associated with the feature/file in a release.	[3]
	Number of active developers (ADEV)	Number of developers who have edited (changed, deleted or added) the feature / file within a release	[3]
	Number of distinct developers (DDEV)	Cumultative number of developers who have edited (changed, deleted or added) the feature / file within a release	[3]
	Experience of all develepoers (EXP)	geometric mean of the experience of all developers who have edited (changed, deleted or added) the feature / file within a release.~ Experience is defined as the sum of the changed, deleted or added lines in the commits associated with the feature / file.	[3]
	Experience of the most involved developers (OEXP)	Experience of the developer who has edited (changed, deleted or added) the feature / file most often within a release. Experience is defined as the sum of changed, deleted, or added lines in the commits associated with the feature/file.	[3]
	Degree of modifications (MODD)	Number of edits (change, removal, extension) of the feature / file within a release.	*
	Scope of modifications (MODS)	Number of edited features / files within a release (feature or file overlapping value). Idea: The more features / files have been edited in a release, the more error-prone they seem to be.	*
Code metrics	Lines of code (NLOC)	Average number of lines of code of the files associated with the feature / ~file within a release.	*
	Cyclomatic Complexity (CYCO)	Average cyclomatic complexity of the files associated with the feature / file within a release.	*
	Number of added lines (ADDL)	Average number of lines of code added to the files associated with the feature / file within a release.	*
	Number of removed lines (REML)	Average number of lines of code deleted from the files associated with the feature / from the file within a release	*

** These values were calculated based on the metadata obtained with PyDriller.*
Feature-level metrics were calculated based on the metadata of the underlying files.