113 Times Tomcat: a dataset

Giuseppe Destefanis[1], Mahir Arzoky[2] Steve Counsell[2], Stephen Swift[2], Marco Ortu[3], Roberto Tonelli[3], and Michele Marchesi[3]

[1] University of Hertfordshire,
g.destefanis@herts.ac.uk
[2] DIEE, University of Cagliari,
|marco.ortu|michele|roberto.tonelli|@diee.unica.it
[3] Brunel University, Uxbridge, UK
|mahir.arzoky|steve.counsell|stephen.swift|@brunel.ac.uk

Abstract. Measuring software to get information about its properties and quality is one of the main issues in modern software engineering. The aim of this paper is to present a dataset of metrics associated to 113 versions of Tomcat. We describe the dataset along with the adopted criteria and the opportunities of research, providing preliminary results. This dataset can enhance the reliability of empirical studies, enabling their reproducibility, reducing their cost, and it can foster further research on software quality and software metrics.

Key words: tomcat, software metrics, software quality

1 Introduction

Software metrics were created in order to improve the software development process, with the goal of measuring and controlling its essential parameters. Even if the meaning is (or was) used in a broad sense, software metrics generally refer to: product, process, resource, or project measurements.

We restrict our attention to the first meaning, dealing with product metrics alone which may be distinguished in size metrics, complexity metrics and quality metrics, generally related to each other. A milestone in the definition of a useful set of software metrics was the work of Chidamber and Kemerer [2], the first trial in addressing the problem of implementing a new suite of metrics for Object Oriented (OO) design.

Measurement is fundamental during each step of software development, and having the possibility to perform analysis on a well-defined corpus of systems is definitely an added value. Reproducibility, reliability and applicability of results or findings can be significantly improved by the use of datasets of software systems. In general, it is possible to find several source of datasets. The International Conference on Predictive Models and Data Analytics in Software Engineering (Promise)[1] strongly encourages researchers *"to publicly share their data in order*

[1] http://promisedata.org

to provide interdisciplinary research between the software engineering and data mining communities", while the Working conference on Mining Software Repositories (MSR)[2], hosts a data-showcase session in which researchers are called on to illustrate and share their datasets.

Modern software systems are made by thousands of classes linked by thousands of dependency relationships and by millions of lines of code. The implications of measurement for the software industry can be very large and can range from software maintainability costs, to defect detection strategies, to resources allocation.

In this work, we present a dataset of 53 metrics taken from 113 versions of Tomcat[3], an open source Java Servlet Container developed by the Apache Software Foundation (from version 3.3.2 to version 8.0.9) and heavily used in software engineering research [14].

The dataset is openly available at the following link *https://bitbucket.org/giuseppedestefanis/tomcatdataset* and contains a SQL file, two SQL views, the source code of all the 113 versions of Tomcat considered and a manual which contains examples on how to use the dataset.

This paper is structured as follows. In Section 2, we illustrate the process of construction and the content of the dataset. In Section 3, we propose several examples on how to use the dataset (along with some of the research opportunities) and present preliminary analysis obtained from it. We then discuss the related work in Section 4 and finally we draw some conclusions (Section 5).

2 Dataset

The goal of this work is to provide a large longitudinal dataset containing software metrics calculated for 113 versions of Tomcat to allow other researchers to perform empirical and predictive studies using a well known and often analyzed system.

Other datasets provided in the past [12, 16] have not contained test classes; they have been considered as outside the system. In this dataset, we voluntarily decided to maintain test classes, since the role that these classes play in the development process and the inextricable relationship they have with production code is an area that has been largely neglected in recent years [15, 17]. Often, the number of test classes is comparable to that (or even exceed that) of production, so the question why they receive such poor attention is highly relevant to industry [4].

The first column of the database contains information about the entities for which metrics are provided. Depending on which kind of metric is considered, it is possible to obtain information about the dimension of a given release of the system, its complexity and OO properties. To calculate these metrics, we used Understand 3.1 (build 766)[4]. We computed all the software metrics available in

[2] `http://msrconf.org`
[3] `http://tomcat.apache.org`
[4] Understand. Scitools.com: `https://scitools.com`

Understand which are categorized in the following groups: Complexity Metrics (e.g., McCabe Cyclomatic)[5], Volume Metrics (e.g, Lines of Code)[6], OO (e.g., Coupling Between Object Classes)[7]. For each system, we provided metrics for 3 different levels of granularity: system, file and class level, considering a total of 61 categories. For example, in the 113 releases of Tomcat, there are a total of 202 *Abstract Classes*, 153656 *Files* (which can contain one or more classes), 122276 *Public Classes*, 150 *Generic Interfaces*, etc.. Table 1 shows some statistics about the different type of *classes* considered (and found in the 113 versions of Tomcat) by the tool. Table 2 shows statistics about the different type of *interfaces*, while Table 3 about the different type of *methods*. Column 1 shows the name of the category, while column 2 the number of instances of that specific category found in our dataset.

Kind	#
Package	295
File	153656
Abstract Class	202
Class	26319
Generic Class	2
Private Abstract Class	351
Private Class	5877
Private Generic Class	13
Private Static Abstract Class	103
Private Static Abstract Generic Class	188
Private Static Class	9593
Private Static Generic Class	122
Protected Class	5943
Protected Generic Class	175
Protected Static Abstract Class	97
Protected Static Abstract Generic Class	79
Protected Static Class	4070
Protected Static Generic Class	124
Public Abstract Class	7947
Public Abstract Generic Class	370
Public Class	122276
Public Generic Class	634
Public Static Abstract Class	207
Public Static Abstract Generic Class	30
Public Static Class	14887
Public Static Generic Class	12
Static Class	2260

Table 1: Class level

[5] `https://scitools.com/support/metrics_list/?metricGroup=complex`
[6] `https://scitools.com/support/metrics_list/?metricGroup=count`
[7] `https://scitools.com/support/metrics_list/?metricGroup=oo`

Kind	#
Interface	180
Generic Interface	150
Public Interface	18702
Public Generic Interface	53
Private Generic Interface	15
Private Interface	57
Protected Interface	63

Table 2: Interface level

Kind	#
Method	16848
Public Generic Method	1252
Public Implicit Method	1
Public Method	1267971
Abstract Generic Method	76
Abstract Method	15184
Private Method	87604
Private Static Generic Method	214
Private Static Method	14238
Public Static Generic Method	41
Public Static Method	80377
Static Method	7326
Protected Method	130711
Protected Static Method	5765
Protected Abstract Method	4285
Public Abstract Generic Method	193
Public Abstract Method	146569

Table 3: Method level

The dataset contains also information about 295 Packages, 12466 Constructors, 253 Enum Type, 918 Public Enum Type, 582 Private Enum Type, 4150 Private Constructor, 2663 Protected Constructor, 188 Protected Enum Type, 1997 Public Annotation, 117417 Public Constructor and 4354 Type Variable. We decided to maintain all categories provided by Understand in the generic database and to provide a set of *sql views* to facilitate the use of the dataset. Namely: *TomcatView*: contains an additional column, called *Version*, which indicates the version the entity belongs to, and *publicClasses*, view which contains only Public Classes from the 113 versions of Tomcat.

3 Querying the Dataset

The SQL query in Figure 1 provides the number of Public Classes per version. The figure has been plotted using R[8]. It is a very simple query which illustrates what kind of information is possible to obtain from the dataset. The figure shows that the number of public classes over time presents a crescent linear trend (diagonal line), and it is also possible to localise points in which the number of public classes drops. Those points could indicate possible major refactoring or restructuring of the system.

In order to better understand the drop of the number of public classes in certain releases, as shown in Figure 1, and to investigate if refactoring has occurred, it could be useful to study the average dimension of the classes per release. Figure 2 shows the results of a SQL query which provides the average dimension (in terms of LOC) of the public classes, per release. It is possible to notice that it is immediately possible to compare the graph in Figure 1 and the graph in Figure 2, and that when the number of public classes decreases, the average dimension of a class increases (and *vice-versa*).

```
SELECT Version, count(kind) as NumberOfPC
FROM TomcatView
where kind='Public Class'
group by Version
```

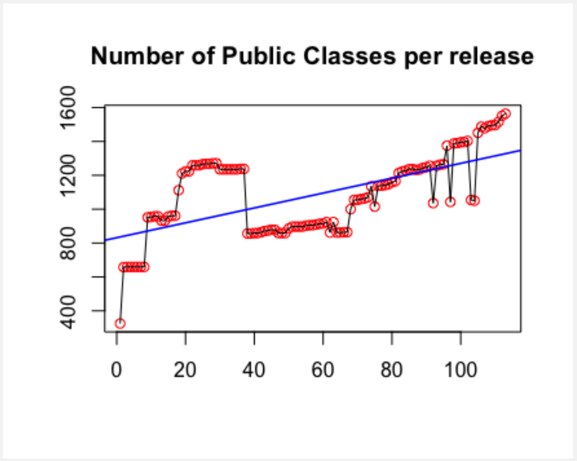

Fig. 1: Number of Public Class per release

This fact reinforces the hypothesis of major refactoring in which several classes are restructured (moving methods for example), and therefore the in-

[8] `https://www.r-project.org`

creased average of LOC per class can find justification in this sense. However, these are only hypotheses which have to (and can) be deeply analysed and confirmed using our dataset. The average does not provide precise information, but in this case it has been used to present the potential of our dataset.

```
SELECT Version, avg(CountLineCode) as AvgLOC
FROM TomcatView
where kind='Public Class'
group by Version
```

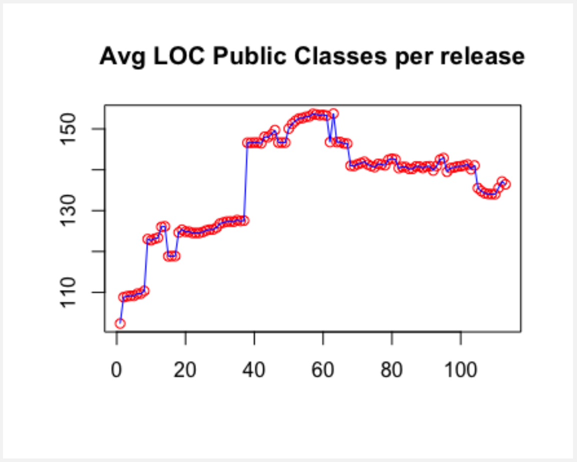

Fig. 2: Average LOC per Public Class per release

```
SELECT Version, count(Name) as NumberOfPC
FROM TomcatView
where kind='Public Class' and Name like '%test%'
group by Version
```

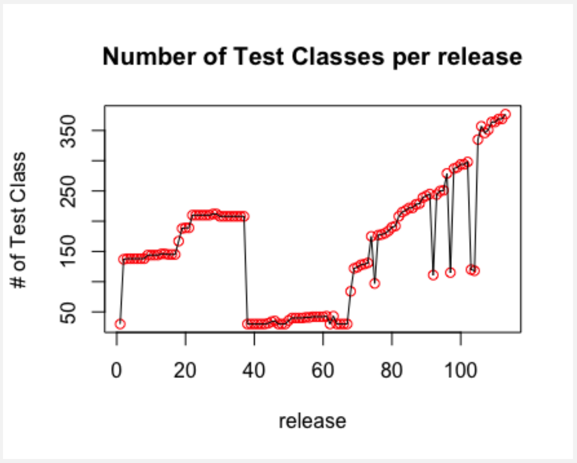

Fig. 3: Number of Test Class per release

The SQL query in Figure 3 provides the number of Public Classes, identified as such if they contained the word *test* in the class name. Another fact which is possible to observe analyzing the dataset and investigating test classes is the alarming difference in number of comment lines between the sets of test and production classes.

Table 4 illustrates the difference in ratios of comment lines between the two types of class, showing mean, median and standard deviation (Std. Dev.) values for the classes in each set. Clearly, from the table, the mean and median values differ significantly. Production classes have a propensity for higher numbers of comment lines than that of test classes, with a mean of over 4 times as great. The median of 0.08 for test classes is dwarfed by the value of 0.39 for production classes. The highest ratio in the set of production classes was 11.50 and, for the set of test classes, just 1.5 (illustrating the gulf).

Class Type	Mean	Median	Std. Dev. #
Test classes	0.14	0.08	0.21
Prod. classes	0.61	0.39	0.79

Table 4: Summary comment ratio statistics

In theory, we might expect both test and production classes to show similar comment line characteristics since they have a synergistic relationship; this does not seem to be the case at first. One suggestion for the discrepancy might be that production classes were simply larger than test classes and that this factor accounts for the higher number of comment lines and higher comment ratio in the former found in Table 4.

Table 5 shows the same values as in Table 4, but for lines of executable code. While the mean values in Table 4 show a relatively large difference, the median values for both sets of classes are not too dissimilar (53 lines of code for test classes versus 60 lines of code for production classes). The difference in the mean values (of 97.76 and 119.36) can be accounted for by a small number of very large production classes skewing the means accordingly. For example, 16 production classes exceeded 1000 lines of code, compared with just 3 for the set of test classes. Clearly, the disparity between the numbers of comment lines (between test and production classes) is not evident from, or explained by, the size of class. Further analysis could be made considering metrics for coupling, cohesion and code nesting.

Class Type	Mean	Median	Std. Dev. #
Test classes	97.76	53	143.59
Prod. classes	119.36	60	179.27

Table 5: Summary lines of code statistics

The metrics can be compared with other measurements of different nature taken from the associated repositories, for example social metrics (as the dataset provided by Ortu et al. [13]) or the corresponding Issue Tracking Systems (e.g. number of issues, defects, etc.), or for patterns detection[5, 3]. Additionally, it could be interesting to investigate if and how metrics change among different components of the same system.

4 Related Work

Reproducibility, reliability and applicability of results or findings can be significantly improved by the use of datasets of software systems. Tomcat has been largely used for empirical software engineering studies [6, 10, 7, 8, 9, 14, 18, 20]. For example, Okutan et al. [11] built a Bayesian network among metrics and defectiveness, to measure which metrics were more important in terms of their effect on defectiveness and to explore the influential relationships among them. The authors used 9 datasets from the teraPromise data repository[9], considering Tomcat among other systems, and showed that RFC, LOC, and LOCQ are more effective on defect proneness. Canfora et al.[1] proposed a multi-objective approach for cross-project defect prediction evaluating 10 datasets, from the teraPromise data repository and considering Tomcat, based on a multi-objective

[9] `http://openscience.us/repo/`

logistic regression model built using a genetic algorithm. The proposed approach allows software engineers to choose predictors achieving a compromise between number of likely defect-prone artifacts and LOC to be analyzed/tested.

Weissgerber et al. [19] used Tomcat, among other systems, to describe three visualization techniques which help to examine how programmers work together, providing very interesting insights on what happens during the development process of a system.

As already stated, few studies have investigated the test-production features of a system's code and the need to explore systems that have evolved is greater than ever and with this longitudinal dataset we encourage works in this direction.

5 Conclusion

In this work, we presented a dataset of 53 metrics associated with 113 versions of Tomcat (from version 3.3.2 to version 8.0.9) along with the source code of all the versions, describing the motivation that led us to build the corpus and providing a description of the dataset and preliminary results obtained querying it. The main goal of the dataset is to provide a benchmark for empirical and predictive longitudinal studies that allows reproducibility of results and lowers the cost of experiments.

References

1. G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. Multi-objective cross-project defect prediction. In Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on, pages 252–261. IEEE, 2013.
2. S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6):476–493, 1994.
3. G. Concas, G. Destefanis, M. Marchesi, M. Ortu, and R. Tonelli. Micro patterns in agile software. In International Conference on Agile Software Development, pages 210–222. Springer Berlin Heidelberg, 2013.
4. S. Counsell, G. Destefanis, X. Liu, S. Eldh, A. Ermedahl, and K. Andersson. Comparing test and production code quality in a large commercial multicore system. In Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on, pages 86–91. IEEE, 2016.
5. G. Destefanis, R. Tonelli, E. Tempero, G. Concas, and M. Marchesi. Micro pattern fault-proneness. In Software engineering and advanced applications (SEAA), 2012 38th EUROMICRO conference on, pages 302–306. IEEE, 2012.
6. S. Hussain, J. Keung, A. A. Khan, and K. E. Bennin. Performance evaluation of ensemble methods for software fault prediction: An experiment. In Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference, pages 91–95. ACM, 2015.
7. L. W. B. X. G. Kaiser and R. Passonneau. Bugminer: Software reliability analysis via data mining of bug reports. delta, 12(10):09–0500, 2011.

8. S. Lal and A. Sureka. Logopt: Static feature extraction from source code for automated catch block logging prediction. In Proceedings of the 9th India Software Engineering Conference, pages 151–155. ACM, 2016.
9. P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam. Empirical evaluation of defect projection models for widely-deployed production software systems. In ACM SIGSOFT Software Engineering Notes, volume 29, pages 263–272. ACM, 2004.
10. M. Monperrus and M. Martinez. Cvs-vintage: A dataset of 14 cvs repositories of java software. 2012.
11. A. Okutan and O. T. Yıldız. Software defect prediction using bayesian networks. Empirical Software Engineering, 19(1):154–181, 2014.
12. M. Orrú, E. Tempero, M. Marchesi, R. Tonelli, and G. Destefanis. A curated benchmark collection of python systems for empirical studies on software engineering. In Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering, page 2. ACM, 2015.
13. M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams. The emotional side of software developers in jira. 2016.
14. B. Robinson and P. Francis. Improving industrial adoption of software engineering research: a comparison of open and closed source software. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, page 21. ACM, 2010.
15. S. H. Tan, D. Marinov, L. Tan, and G. T. Leavens. @ tcomment: Testing javadoc comments to detect comment-code inconsistencies. In Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, pages 260–269. IEEE, 2012.
16. E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. The qualitas corpus: A curated collection of java code for empirical studies. In Software Engineering Conference (APSEC), 2010 17th Asia Pacific, pages 336–345. IEEE, 2010.
17. A. Van Deursen, L. Moonen, A. van den Bergh, and G. Kok. Refactoring test code. In Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001), pages 92–95, 2001.
18. F. Van Rysselberghe and S. Demeyer. Mining version control systems for facs (frequently applied changes). In 26th International Conference on Software Engineering, Edinburgh, Scotland, pages 48–52, 2004.
19. P. Weissgerber, M. Pohl, and M. Burch. Visual data mining in software archives to detect how developers work together. In Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on, pages 9–9. IEEE, 2007.
20. T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pages 91–100. ACM, 2009.