DVAD14 Automated Software Engineering - Replication Project A Study on Architectural Smells Prediction

Noah Bühlmann
noah.buehlmann@students.unibe.ch
University of Bern
Bern, Switzerland

Roland Widmer
roland.widmer@students.unibe.ch
University of Bern
Bern, Switzerland

Hugo Baptista
fc54442@alunos.fc.ul.pt
University of Lisbon
Lisbon, Portugal

ABSTRACT

Architectural smells indicate an architecture that may result in poor software quality. Thus, it is helpful to examine the development of architectural smells in order to prevent the negative impact on software quality. This work is a replication of the paper "A Study on Architectural Smells Prediction" by Arcelli Fontana et al. [2], who examined whether the presence of architectural smells in earlier versions can be used to predict the presence in future versions. By analyzing 2.5 years of the commit history of another previously unexamined project we were able to obtain similar results as in the original paper. By applying four different machine learning algorithms, we could train models that can reliably predict the existence of architectural smells.

CCS CONCEPTS

• **Software and its engineering** → **Software development techniques**; • **Computer systems organization** → Architectures.

KEYWORDS

software, architecture, architectural smells, machine learning

1 INTRODUCTION

Automated software engineering revolves around the idea of using automation techniques to speed up repetitive tasks and/or utilizing machine learning in order to find solutions and patters in data pools too large or complex for humans to do manually. In this project we are using said tools to detect and predict architectural smells (AS) in Java projects.

AS happen when the project, at its genesis and develoment, implements bad practices or does not abide by certain architectural principles, that are used to make sure a number of weak points are not created and therefore do not generate issues in the future. (Weak points like cyclic dependencies, per example, where a given amount of objects depend on each other in ways that if one fails everything fails, this could obviously lead to major faults/failures in the system on a long-term basis [1]).

Since we wanted to study the evolution of the smells in a project's history that meant analysing thousands of commits, in order to do that and to agregate all the gathered data we used automated scripts. As for getting the results and making the prediction we used several machine learning tools.

The rest of this paper is organized as follows: in section 2 we introduce related work in the field of architectural smells detection and describe in particular detail the existing work we are replicating; in section 3 we explain the replication procedure and the differences to the original work; in section 4 we present the main results of our replication; and finally, in section 5 we outline the threats to validity of the work.

2 RELATED WORK

The work in this replication project is mainly based on the paper "A study on Architecural Smells Prediction" by Arcelli Fontana et al. [2]. For that reason we describe their methods and results in detail in subsection 2.1.

There also exists of a large number of other related work in the field of history-based analysis of software projects. For example, Taba et al. [10] explored the presence of antipatterns for bug prediction by studying mutliple versions of Eclipse and ArgoUML. Furthermore, in the area of bug and smell prediction Maneerat and Muenchaisri [7] used 7 machine learning techniques to show that bad smell prediction from software design models can predict bad smells earlier. Previous research also targeted dependencies, a common cause for AS. Oyetoyan et al. [9] empirically showed that classes involving circular depencies are more change-prone and Diaz-Pace et al. [4] tried to predict dependency-based AS using structural system characteristics.

2.1 A Study on Architectural Smells Prediction [2]

2.1.1 Architectural Smells Detection. For the detection of AS the authors used the tool Arcan [5]. This tool takes as input a compiled Java project and models its structural dependencies in a graph model. By applying detection algorithms to this graph Arcan is then able to extract metrics about the packages and classes of the projects. Using those metrics Arcan detectes four architectural smells:

- Unstable Dependency (UD)
- Hub-Like Dependency (HL)
- Cyclic Dependency (CD)
- Implicit Cross Package Dependency (ICPD)

For CD smells Arcan can further detect the shapes as proposed by Al-Mutawa et. al [1]: tiny, clique, circle, chain and star shapes. For the ICPD smell, in contrast to the other three smells, the history of the project is needed for detection and not only a single version snapshot.

The authors have chosen those four architectural smells, because they believe they represent critical problems. In particular, the Cyclic Dependency smell is one of the most common smells and considered the most critical by developers [8].

2.1.2 Case study setup. Arcelli Fontana et al. wanted to answer the following research question:

RQ: *How well does the presence of architectural smells in the project's history support the prediction of architectural smells in the future?*

In order to so, they analyzed four projects chosen from Github[1]: JGit, JUnit4, Commons-Math and Apache Tomcat. The main features of those projects are summarised in the left part of table 1. These four projects were selected because they meet the following criteria, that the authors have imposed:

- Written in the Java programming language
- More than 5 years of activity
- At least 2000 commits
- At least 20 major releases

As their dataset a collection of data points for each (version, package) combination for each software project was used. The authors decided to work on package level because not all AS are defined on class level. For each data point, they represented each of the four AS as a binary feature, with value 1 if the package was involved in an AS in that version and 0 otherwise. Furthermore for CD they also distinguished between *tiny, star, chain, circle and clique* features. All those AS features were used as targets of the predicitions, as well as features for the training phase.

To represent AS in past versions in such a way that it enables the application of prediction models on the data, the authors chose the concept of *lagging*. They chose to use the past 12 versions ("lags") to predict the presence of AS in the next version, i.e. they added to each row (package, version) the AS features of (package, version - 1), (package, version - 2), . . . , (package, version - 12). In addition to the lagging, the authors decided to aggregate the whole dataset by month, in order to account for the fact that architectural changes are rather long-term than short-term and in order to be able to include more data into the study while keeping computational cost reasonably low. Two more preprocessing steps were applied. First, the authors removed features with variance near to zero. Secondly, SMOTE [3] sampling was applied to counter class imbalance. After this pre-processing phase, the remaining main part of the analysis was carried out as a standard supervised learning task.

For the machine learning the R implementations of five common models were used: Naïve Bayes (NB), Decision Trees (C5.0), Random Forests (RF) and Support Vector Machines (SVM). For the latter they tested two types *svmRadial* (SR) and *svmLinear* (SL). In order to evaluate the different models a standard repeated k-fold cross validation procedure was carried out, with 10 repetitions and 10 folds. The performance was mainly judged by the common metrics *Accuracy* and *F-Measure*. The Wilcoxon signed rank test [6] was applied to the F-measure scores. Naturally only performances of different models on the same dataset (software project) were compared. To judge if a difference in the average F-Measure is significant, i.e. if the performance of the different models is distinguishable, a significance level of $\alpha < 0.05$ was used.

2.1.3 *Results on prediction.*

The computed F-measure and accuracy metrics are visible in the upper grey-colored part of table 2. The values higher or equal to 0.6 are highlighted in bold, indicating the best performances. The authors report that the performance is generally high, with very few exceptions. The worst performing model is NB prediction, which has much lower accuracy than the other classifiers throughout all projects. The best classifiers are SVM Linear and SVM Radial, which is also confirmed by significant p-values from the Wilcoxon test (p-values between 0 and 0.03).

When it comes to the different AS the authors note that the highest performance is obtained in the prediction of the CD smell and its shapes. They also point out, however, that this result could be influenced by class imbalance which is a valid concern in classification tasks.

2.1.4 *Conclusion and future developments.*

Arcelli Fontana et al. conclude their paper by answering their research question. They state that the prediction performance is high or very high and thus historical AS information can be used to predict the presence of AS in the future. They point out that developers and maintainers of software projects should pay special attention to CD as they are a good predictor for CD smells and should be removed as soon as the AS is introduced and detected.

For future work the authors hope that their study can be extended by analysing more projects. This is exactly what this replication project is trying to achieve. Furthermore they suggest to expand the research to more and different AS and a larger set of lag settings in order to get a more precise overview of the prediciton performance of the different models.

3 REPLICATION PROCEDURE

If not described otherwise, the same steps, data format and tools were used for this replication, that the authors of the original paper used. The AS are detected using Arcan. Arcan takes a compiled Java program as input and returns a set of csv files with metrics about detected AS. After some merging and preprocessing, this data can be used to train the classifier as in the original paper.

3.1 Project selection

Some more technical and practical requirements were added in addition to the project selection criteria imposed by the original authors and described in section 2.1:

- Java project: Java binaries are required for Arcan to detect AS.
- Open source project or downloadable builds per commit: We need to run Arcan for many different commits.
- Straightforward build: The build process should be simple and reasonably fast. Otherwise, the data collection would take a long time, as we need to build for every commit. To keep the data collection simple, the build process should also be similiar and stable over the observed time period.
- Enough commits and data: To collect enough data, we need may commits and multiple years of activity. For this point, we used the same requirements as the original paper (five years of activity, at least 2000 commits, and at least 20 major releases).

There are various projects available that meet those requirements. We chose JabRef[2], an open-sourced, cross-platform citation and reference management software. It fulfills all requirements and was already known to us from previous research projects. Some general metrics of JabRef, compared with the projects analysed in

[1]https://github.com

[2]https://github.com/JabRef/jabref

Table 1: Selected projects
Columns 1-4 by Arcelli Fontana et al. [2]

	JGit	JUnit4	Commons-Math	Apache Tomcat	JabRef
Domain	library	library	library	app server	desktop app
# Github releases	83	20	65	127	7
# Versions (Commits)	4840	2187	6286	6853	2415
Date Start	2009-09-29	2000-12-03	2003-05-03	2006-03-28	2019-10-07
Duration (years)	8	17	15	6	2.38
# Classes (first commit)	454	60	9	83	1579
# Packages (first commit)	20	10	1	77	150
# Classes (last commit)	1315	445	1393	2218	1894
# Packages (last commit)	45	32	20	141	188

the initial study, are shown in table 1. It should be noted here that all metrics in table 1 denote the actual "observed" time interval, i.e. the data that was used for the analysis. In the case of JGit, JUnit4, Commons-Math and Apache Tomcat this is also the full history of the project itself.

3.2 Data collection

We developed a PowerShell script that checks out Git commits in reverse chronological order. Then, the build process is started. JabRef uses Gradle[3], a build automation tool. As the required Gradle and Java versions have changed over time, the script has to choose the correct version using a mapping between Gradle and Java version to ensure compatibility. The result of the build is a folder of Java binaries that is used as input for Arcan. If Arcan terminates and outputs the result, the script adds some additional columns to the data - namely the version (commit hash) and the commit timestamp. Eventually, the script checks out the next commit and continues for a fixed amount of iterations. The result of this script are enriched Arcan outputs. As we run Arcan for every commit, we have an output file per commit and per AS. To get the same data structure as in the original paper, we used a second script. The script merges the files per commit to a single file per AS. As we added the commit hash as a column, we are still able to infer the commit hash of a given data point.

3.3 Preprocessing

The authors of the original paper provide a R script to preprocess merged Arcan output. With some minor bug-fixing adjustments we were able to preprocess our data. The script merges the different files (per AS) to a single file. Moreover, it aggregates the datapoints to get a single datapoint per month. The commit date column is used to assign every commit a month. The chosen aggregation function depends on the given metric. Mean, sum, minimum and maximum functions are supported.

[3]https://gradle.org

3.4 Classifier training

As in the previous section, we reused the R script from the original study for the machine learning as well. We had to do some adjustments which are discussed in the following section. The R script reads the data and trains classifiers using different algorithms. In the end, the confusion matrices and other performance measures are returned. Those results are discussed and compared to the results of the original paper in section 4.

3.5 Differences

In our replication, we encountered some issues and had to do some adjustments compared to the original paper:

- IXPD: The used version of Arcan does not support this metric. Therefore, we were not able to collect any data and ignored this metric.
- Naïve Bayes: With our setup, we were not able to train the classifier using the Naïve Bayes algorithm. As this algorithm had the worst performance of all examined algorithms in the original paper, we omitted this algorithm.
- HL: The HL metric is mostly 0 for JabRef, thus this feature has a near-zero variance. Therefore, it is not relevant for a classifier and our models cannot predict HL dependencies. The authors of the original paper also ommited this metric for all projects, except for JGit, presuambly for the same reasons.

3.6 Replication package

We provide a replication package[4] that can be used to replicate all the steps that we carried out and reproduce our results. The intermediate and final results of our analysis are also provided online.

[4]https://github.com/TheNCuber/dvad14-smelly-replication/

4 RESULTS AND FUTURE WORK

The results of this replication project are reported in table 2. The grey parts indicate the exisiting results by Arcelli Fontana et al. [2], while the blue parts are our contributions. In compliance with the conventions by the original authors, results greater or equal 0.6 are indicated in bold. Despite the smaller sample size, our results with JabRef are very much in line with the results of the original study. The prediction of the CD smell is highly accurate throughout all models, while the prediction of UD proved again to be challenging for non-SVM models.

The results of the Wilcoxon signed rank test [6] applied to our own results let us also conclude, that the SL classifier performs significantly better (p value = 0) than the C5.0 and RF models. Not enough evidence is present to infer the same for the SR classifier as the original authors were able to.

Table 2: Accuracy and F-measure results of ML models Grey results by Arcelli Fontana et al. [2]

		Accuracy					F-Measure				
	AS	C5.0	NB	RF	SL	SR	C5.0	NB	RF	SL	SR
Commons-Math	CD	**.995**	**.981**	**.995**	**.858**	**.823**	**.997**	**.989**	**.997**	.537	.421
	CD-chain	**.837**	**.624**	**.831**	**.765**	**.732**	**.788**	**.643**	**.783**	**.804**	**.776**
	CD-clique	**.945**	**.764**	**.942**	**.928**	**.920**	**.767**	.384	**.762**	**.960**	**.956**
	CD-star	**.897**	**.610**	**.894**	**.922**	**.922**	.553	.269	.557	**.958**	**.958**
	CD-tiny	**.845**	**.666**	**.846**	**.665**	**.676**	**.816**	**.694**	**.815**	**.706**	**.716**
	UD	**.669**	.504	.597	.571	**.699**	.039	.419	.221	**.685**	**.779**
	ICPD	**.711**	**.768**	**.712**	**.887**	**.862**	.235	.211	.255	**.938**	**.924**
JGit	CD	**.996**	**.986**	**.996**	**.999**	**.999**	**.998**	**.992**	**.998**	**.999**	**.999**
	CD-chain	**.833**	**.665**	**.846**	**.999**	**.993**	**.757**	.437	**.775**	**.999**	**.990**
	CD-circle	**.765**	**.630**	**.782**	**.999**	**.995**	**.756**	**.713**	**.769**	**.999**	**.994**
	CD-clique	**.953**	**.799**	**.955**	**.999**	**.997**	**.849**	.500	**.855**	**.999**	**.990**
	CD-star	**.785**	**.646**	**.800**	**.999**	**.994**	**.683**	.443	**.726**	**.999**	**.992**
	HL	**.684**	**.747**	**.753**	**.999**	**.985**	.175	.211	.121	**.999**	**.827**
	UD	.585	**.638**	**.732**	**.999**	**.990**	.233	.255	.132	**.999**	**.959**
	ICPD	**.671**	.590	**.710**	**.999**	**.981**	.252	.305	.232	**.999**	**.950**
JUnit	CD	**.998**	**.896**	**.999**	**.999**	**.958**	**.998**	**.871**	**.999**	**.999**	**.977**
	CD-chain	**.998**	**.899**	**.998**	**.999**	**.847**	**.998**	**.884**	**.998**	**.999**	**.875**
	CD-star	**.994**	**.753**	**.993**	**.999**	**.903**	**.995**	**.793**	**.993**	**.999**	**.622**
	CD-tiny	**.997**	**.884**	**.997**	**.999**	**.857**	**.997**	**.893**	**.997**	**.999**	**.724**
	UD	**.939**	.587	**.937**	**.999**	**.823**	**.943**	**.715**	**.941**	**.999**	.524
	ICPD	**.997**	**.959**	**.997**	**.999**	**.874**	**.996**	**.954**	**.997**	**.999**	**.898**
Tomcat	CD	**.999**	**.977**	**.999**	**.999**	**.999**	**.999**	**.986**	**.999**	**.999**	**.999**
	CD-chain	**.996**	**.913**	**.997**	**.999**	**.998**	**.997**	**.933**	**.998**	**.999**	**.999**
	CD-clique	**.996**	**.793**	**.996**	**.999**	**.999**	**.982**	.409	**.982**	**.999**	**.995**
	CD-star	**.989**	**.760**	**.986**	**.999**	**.995**	**.980**	**.666**	**.976**	**.999**	**.990**
	ICPD	**.750**	**.620**	**.723**	**.999**	**.996**	.292	.269	.289	**.999**	**.978**
JabRef	CD	**.999**	x	**.999**	**.999**	**.997**	**.999**	x	**.999**	**.999**	**.999**
	CD-chain	**.970**	x	**.971**	**.999**	**.956**	**.975**	x	**.976**	**.999**	**.966**
	CD-circle	**.941**	x	**.942**	**.999**	**.958**	**.884**	x	**.885**	**.999**	**.908**
	CD-clique	**.990**	x	**.991**	**.999**	**.974**	**.950**	x	**.952**	**.999**	**.845**
	CD-star	**.979**	x	**.981**	**.999**	**.959**	**.981**	x	**.982**	**.999**	**.963**
	CD-tiny	**.961**	x	**.957**	**.999**	**.969**	**.959**	x	**.954**	**.999**	**.967**
	UD	**.622**	x	**.660**	**.999**	**.923**	.552	x	.541	**.999**	**.902**

4.1 Future work

The work carried out for this replication, as well as the original study, could be expanded in multiple directions in the future.

Analogous to this replication project more software projects could be analysed. In particular, it would be interesting to include software that is based on another programming language than Java and compare the results. This would of course require the usage of a different tool than Arcan for AS detection.

When using different tools anyway, it could be also worthwhile to explore different AS than the ones used in this study and see if they can also be used to predict AS in the future.

Another point that should be explored in the future is the impact of different data aggregation strategies. Both this replication and the original study aggregated the data per months, but it remains unclear if different aggregation intervals would yield better or worse results.

Finally, it was already pointed out by Arcelli Fontana et al. [2] that they are interested to find out if the presence of AS in the projects evolution can be used to predict software changes and vice-versa.

5 THREATS TO VALIDITY

Since this is a replication project the threats to validity are tied in a very significant way to the ones outlined by the authors of the original paper [2], the most relevant of them being the possibility of data distortions caused by the aggregation process. We also aggregated the collected commit data on a monthly basis, this means that commits are combined into a single monthly datapoint. This could lead to "masking effects" if the number of commits varies greatly between months.

A minor threat that was introduced by our data collection process is the fact that we neglected timezone information in the commit history, because the exisiting R code required timezone-free timestamps. This could lead to the problem that a commit by someone with a very different timezone that was made at the very beginning or end of the month could end up counting for the wrong monthly aggregated datapoint.

Other possible threats outlined in the original work are the accuracy of the AS detection tool in use - Arcan - as well as the limited number of analysed projects, which our results are contributing towards, and the fact that only Java based projects were analysed so far which could lead to selection bias.

ACKNOWLEDGMENTS

The following list shows the individual contributions to this project.

- Arcan: Noah Bühlmann, Hugo Baptista
- Gradle builds: Hugo Baptista
- main data collection script (checkout commits, build, run Arcan and merge results): Noah Bühlmann
- preprocessing R script: Roland Widmer
- machine learning R script: Noah Bühlmann, Roland Widmer
- final report: Noah Bühlmann (2, 4, replication package), Hugo Baptista (1, 5), Roland Widmer (3, abstract, acknowledgments)

We thank Sebastian Herold for supervising this project and his valuable comments.

REFERENCES

[1] Hussain A. Al-Mutawa, Jens Dietrich, Stephen Marsland, and Catherine McCartin. 2014. On the Shape of Circular Dependencies in Java Programs. In *2014 23rd Australian Software Engineering Conference*. IEEE, Milsons Point, NSW, Australia, 48–57. https://doi.org/10.1109/ASWEC.2014.15

[2] Francesca Arcelli Fontana, Paris Avgeriou, Ilaria Pigazzini, and Riccardo Roveda. 2019. A Study on Architectural Smells Prediction. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, Kallithea-Chalkidiki, Greece, 333–337. https://doi.org/10.1109/SEAA.2019.00057

[3] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip. Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16 (June 2002), 321–357. https://doi.org/10.1613/jair.953

[4] Jorge Andres Diaz-Pace, Antonela Tommasel, and Daniela Godoy. 2018. Towards Anticipation of Architectural Smells Using Link Prediction Techniques. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, Madrid, 62–71. https://doi.org/10.1109/SCAM.2018.00015

[5] Francesca Arcelli Fontana, Ilaria Pigazzini, Riccardo Roveda, Damian Tamburri, Marco Zanoni, and Elisabetta Di Nitto. 2017. Arcan: A Tool for Architectural Smells Detection. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, Gothenburg, 282–285. https://doi.org/10.1109/ICSAW.2017.16

[6] Myles Hollander, Douglas A. Wolfe, and Eric Chicken. 2015. *Nonparametric Statistical Methods* (first ed.). Wiley. https://doi.org/10.1002/9781119196037

[7] Nakarin Maneerat and Pomsiri Muenchaisri. 2011. Bad-Smell Prediction from Software Design Model Using Machine Learning Techniques. In *2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE, Nakhon Pathom, Thailand, 331–336. https://doi.org/10.1109/JCSSE.2011.5930143

[8] Antonio Martini, Francesca Arcelli Fontana, Andrea Biaggi, and Riccardo Roveda. 2018. Identifying and Prioritizing Architectural Debt Through Architectural Smells: A Case Study in a Large Software Company. In *Software Architecture*, Carlos E. Cuesta, David Garlan, and Jennifer Pérez (Eds.). Vol. 11048. Springer International Publishing, Cham, 320–335. https://doi.org/10.1007/978-3-030-00761-4_21

[9] Tosin Daniel Oyetoyan, Jean-Remy Falleri, Jens Dietrich, and Kamil Jezek. 2015. Circular Dependencies and Change-Proneness: An Empirical Study. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, Montreal, QC, Canada, 241–250. https://doi.org/10.1109/SANER.2015.7081834

[10] Seyyed Ehsan Salamati Taba, Foutse Khomh, Ying Zou, Ahmed E. Hassan, and Meiyappan Nagappan. 2013. Predicting Bugs Using Antipatterns. In *2013 IEEE International Conference on Software Maintenance*. IEEE, Eindhoven, Netherlands, 270–279. https://doi.org/10.1109/ICSM.2013.38