Novel Insights on Cross Project Fault Prediction Applied to Automotive Software

Harald Altinger[1(✉)], Steffen Herbold[2], Jens Grabowski[2], and Franz Wotawa[3]

[1] Audi Electronics Venture GmbH, 85080 Gaimersheim, Germany
harald.altinger@audi.de
[2] Institute of Computer Science, University of Göttingen, 37077 Göttingen, Germany
{herbold,grabowski}@cs.uni-goettingen.de
[3] Institute for Software Technology, Graz University of Technology, 8010 Graz, Austria
wotawa@ist.tugraz.at

Abstract. Defect prediction is a powerful tool that greatly helps focusing quality assurance efforts during development. In the case of the availability of fault data from a particular context, there are different ways of using such fault predictions in practice. Companies like Google, Bell Labs and Cisco make use of fault prediction, whereas its use within automotive industry has not yet gained a lot of attraction, although, modern cars require a huge amount of software to operate. In this paper, we want to contribute the adoption of fault prediction techniques for automotive software projects. Hereby we rely on a publicly available data set comprising fault data from three automotive software projects. When learning a fault prediction model from the data of one particular project, we achieve a remarkably high and nearly perfect prediction performance for the same project. However, when applying a cross-project prediction we obtain rather poor results. These results are rather surprising, because of the fact that the underlying projects are as similar as two distinct projects can possibly be within a certain application context. Therefore we investigate the reasons behind this observation through correlation and factor analyses techniques. We further report the obtained findings and discuss the consequences for future applications of Cross-Project Fault Prediction (CPFP) in the domain of automotive software.

Keywords: Project fault prediction · Cross project fault prediction · Automotive · Principal component analysis

1 Introduction

A modern day premium car like the 2016s Audi Q7 contains up to 90 Electronic Control Unit (ECU)s, 11 different communication networks (Controller Area Network (CAN), FlexRay, etc.) and a wide range of Advanced Driver Assistance Systems (ADAS) that are all realized using software. An analysis carried out by Broy [4] states that software consumes up to 40 % of a cars development

K. El-Fakih et al. (Eds.): ICTSS 2015, LNCS 9447, pp. 141–157, 2015.
DOI: 10.1007/978-3-319-25945-1_9

budget. Software used in cars has up to 10 million Lines Of Code (LOC). Assuring quality of software thus is a major challenge as its complexity is still rising and the currently used testing approaches like Hardware in the Loop (HiL) tests might not scale well. In the automotive industry software engineering follows the W-development process [15], where testing is a core part of every development stage. Predictions about the locations of faults would be a powerful tool to support this process. This paper uses metrics data obtained when carrying out projects for making fault predictions.

In this paper, we contribute to this goal of having tools for fault prediction in the automotive industry. In this work, we consider both the performance of prediction within the context of a project, i.e., fault predictions based on earlier revisions of a project, as well as cross-project predictions, i.e., fault predictions based data from other projects. Considering cross-project predictions is required because empirical data of a project is of course missing when starting the project. In our empirical analysis we focus on the question of whether a strict development process applied for safety and security relevant software modules actually supports fault prediction performance due to restrictive policies. The results obtained do not indicate that a strict development process has an influence on prediction performance. Hence, we also performed an in-depth analysis of possible reasons behind it and discuss this analysis in detail in this paper.

The main contributions of this paper are, an empirical study on within-project and CPFP for safety-critical automotive software as well as an in-depth analysis of the obtained results.

The remainder of this paper is structured as follows. We discuss related work in Sect. 2. Then we present our fault prediction study in Sect. 3. Afterwards, we perform an in-depth analysis on the causes behind the obtained prediction performance in Sect. 4. In addition we discuss threats to its validity in Sect. 5 and finally conclude the paper in Sect. 6.

2 Related Work

Altinger et al. [2] identified Matlab Simulink as the major development environment in the automotive industry, which is often combined with model-based testing and unit testing. In the testing stage there is an equal distribution between Model in the Loop (MiL), Software in the Loop (SiL) and HiL where more than 50 % of the developers are using white or grey box testing techniques. The applicability of Software Fault Prediction (SFP) techniques in such an automotive software development was analysed by Rakesh et al. [23]. They surveyed eight different methods and their best operation time along the life cycle as well as their required input data and their potential to enhance software quality. In comparison to Rakesh et al., the projects we consider in this paper follow a different development process. Our underlying process comprises at least 4 Releases. One represents an *interface freeze*, where external interfaces (CAN, Application Programming Interface (API), methods, etc.) need to be fixed and are not allowed to be changed anymore. *feature freeze*, where all functions need

to be implemented, but might not run fully. They just need to be call-able from external functions. The last release prior to the shipment with the car to customers is *100 % Software* which means that it is the last delivery. After this point, there will be only bug fixes and parameter optimisations, but no feature enhancements.

2.1 Industrial Fault Prediction

SFP is still a very active research field where many datasets are available and studies have been performed. Since we focus on the industrial application of fault prediction in the automotive industry, we only discuss related work in the context of papers dealing with industrial applications of fault prediction methods. For surveys and benchmarks for fault prediction in general we refer the interested reader to, e.g., [6]. The available industrial reports are mainly from Bell labs reporting on their obtained experiences, e.g., Bell et al. [3], Ostrand et al. [21] and et al. [22]. There the authors report a high prediction performance of up to 80 % true positives using three different telecom software products which were developed at AT&T. The authors claim industrial developers benefit from their achieved performance. All reports benefit from a strict bug reporting policy. The underlying fault prediction approach is mainly based on recent changes to files. In contrast to this paper, we consider a machine learning based approach that relies on metrics that can be easily obtained from the software during development.

2.2 Cross-Project Fault Prediction

To overcome the lack of historical data in the early stages of a project's development CPFP uses data from other projects to train fault prediction models. Due to the heterogeneity between different projects, this is a difficult problem. Zimmermann et al. [29] performed a cross-project study on industrial code from Microsoft as well as popular open source software. They state that only 3.4 % of their analysed cross-project predictions achieve more than 0.75 recall, precision and f-measure. However, they demonstrated that whenever project factors are included in the selection of training data during the learning of a decision tree, the likelihood of good predictions can be improved.

Throughout the last years various approaches for improving fault predictions were proposed. In this work, we apply the k-nearest neighbour approach for data selection by Turhan et al. [26] and the normalization approaches by Watanabe et al. [27] and Nam et al. [20]. Other approaches that could be applied are, e.g., a data weighting technique by Ma et al. [18] or transformation of the data according to Camargo Cruz et al. [5]. Moreover, approaches based on the selection of appropriate projects for training, like proposed by He et al. [12] and Herbold [14] are not considered, because we only rely on a setting comprising two projects, where a further selection is obviously not possible.

3 Defect Prediction Case Study

In this section, we present a case study in which we tackle the following research question:

RQ1: Can CPFP be applied in projects using automatically generated code and restrictive coding standards like MISRA [25] which was developed for the same target platform?

This research question **RQ1** is answered by the evaluation of two hypotheses that we initially suspect to be true based on our experience obtained from previous research:

H1: Within-project fault prediction can be applied successfully to automatically generated code.
H2: CPFP is enabled by auto-generated code within a restrictive setting.

Our rational for **H1** is that there are many examples for successful fault predictions in a within-project setting reported in the literature (see, e.g., Catal et al. [6] for an overview). We expect that this is also true for the generated code.

Our rational for **H2** is that differences between metrics due to developer characteristics, etc. can be excluded. Moreover, we expect that the rules used by code generators lead to repetitive patterns in the generated source code. This, in turn, should lead to patterns in the metric values, which should lead to strong correlations between the metric values, even between projects.

Rational for **H2** is that differences between the project context are one of the conjectured greatest threats to CPFP. Zimmermann et al. [29] showed that with a simple decision tree based on context factors the performance of predictions can be greatly improved. In a more recent publication Zhang et al. [28] again demonstrated the power of using context factors. The context factors of the projects we consider are nearly identical. They were developed by the same company, using the same development process, with source code automatically generated with the same code generator following a strict coding standard. This removes a lot of project-specific and developer-specific noise from the data.

3.1 Evaluation Criteria

To evaluate the performance of the fault prediction models, we use the following metrics.

$$recall = \frac{tp}{tp + fn} \tag{1}$$

$$precision = \frac{tp}{tp + fp} \tag{2}$$

$$F\text{-}measure = 2 \cdot \frac{recall \cdot precision}{recall + precision} \tag{3}$$

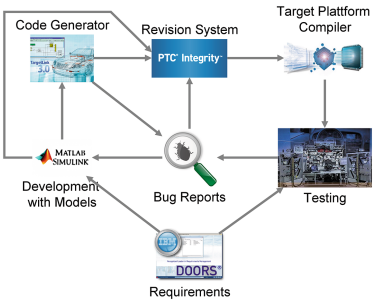


Fig. 1. Workflow during software development

In the above definition, tp, respectively tn are the number of true positive respectively negative predictions, fp, respectively fn are the number of false positive respectively negative predictions. The *recall* measures how many of the existing faults are found. The *precision* measures how many of the found results are actually faults. The *F-measure* is the harmonic mean between *precision* and *recall*. The *error* measures the overall rate of misclassification.

3.2 Data Description

We use the data from a publicity available dataset released by Altinger et al. [1]. The dataset contains fault data of three automotive projects for proprietary reasons simply refereed as A, K, and L. The size of the projects ranges from 10.000 LOC to 36.500 LOC. Two of the three projects are safety relevant, which means that the testing effort had been very high. A special attribute of the data is that the software was not developed by writing source code but by creating Matlab/Simulink models. The source code is then generated automatically using the dSpace TargetLink code generator that fulfils the MISRA [25] guidelines. The workflow during the software development cycle is visualized in Fig. 1. The revisions from all development tools have been analysed and tested. Therefore, the projects stay with the same software during their whole life cycle. As the three projects are from the same time scale, the versions and settings are identical.

Whereas the dataset contains three projects, we only consider the A and the K project in this paper. The reason for this is that the third project L has a very low fault rate, with only three unique faults detected during development. This fault rate is too low to be used in a machine learning approach for fault prediction.

3.3 Defect Prediction Models

In this paper, we use one classification model for within-project predictions and six classification models for cross-project predictions. With the first model, we look at the performance of within-project predictions for baseline comparison

reasons. For the cross-project predictions, we predict the faults in the A using the data from K and the faults in K using the data from A. The classification models we use are the following:

- WP: a within-project fault prediction model. Since the data is ordered by time, we use the data from the oldest 50 % of revisions for training and the remaining 50 % of revisions for evaluation of predictions.
- KNN: cross-project prediction with the k-nearest neighbour approach for data selection introduced by Turhan et al. [26]. We use $k = 10$, which is the same as for the original study. This means, for every entity in the target project, we select the 10 closest entities in the training project for our training data.
- N1: cross-project prediction with min-max normalization [17] of the training and target data separately to the interval [0,1], i.e.,

$$\hat{m}_i(s) = \frac{m_i(s) - \min_{s' \in S} m_i(s')}{\max_{s' \in S} m_i(s') - \min_{s' \in S} m_i(s')}.$$

  This approach for normalization is quite common and, e.g., used by [12–14,18,20,27].
- N2: cross-project prediction with z-score normalization [17], which transforms the training and the target data separately such that the mean value and the standard deviation to one, see e.g. Nam et al. [20], i.e.,

$$\hat{m}_i(s) = \frac{m_i(s) - mean(m_i(S))}{std(m_i(S))}.$$

- N4:[1] Cross-project prediction with z-score normalization of the training and target data, both based on the mean and standard deviation of the target data after Nam et al. [20], i.e.,

$$\hat{m}_i(s) = \frac{m_i(s) - mean(m_i(S^*))}{std(m_i(S^*))}.$$

- N5: cross-project prediction with normalization of data according to the mean standardization proposed by Watanabe et al. [27], i.e.,

$$\hat{m}_i(s) = \frac{m_i(s) \cdot mean(m_i(S^*))}{mean(m_i(S))}.$$

For the training of all classification models, we used under sampling [8] to treat the bias towards non-fault-prone classifications due to the small number of fault-prone entities in the data sets. As classifier, we used a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel [24], one of the overall best performing classifiers from the machine learning literature [10]. Please note that we did not use a cross-project model without normalization. The reasons for this are that we used a SVM as classifier and SVMs often perform poorly if no scaling or normalization is used [10].

[1] We use N4 instead of N3 to be consistent to the naming of Nam et al. [20].

3.4 Prediction Performance

In the following, we report the results obtained for fault prediction. Table 1 lists the values for recall, precision, and F-measure achieved with the prediction models. The first column contains the within-project predictions, the other columns the results for the cross-project predictions with the various transfer learning techniques.

Table 1. Results achieved with the various classification models on the data sets. Abbreviations see Sect. 3.3

		WP	KNN	N1	N2	N4	N5
Recall	A	0.59	-	-	-	-	-
	K → A	-	0.48	0.48	0.48	0.48	0.48
	K	0.89	-	-	-	-	-
	A → K	-	0.75	0.75	0.75	0.75	0.69
Precision	A	0.16	-	-	-	-	-
	K → A	-	0.19	0.19	0.19	0.19	0.18
	K	0.21	-	-	-	-	-
	A → K	-	0.20	0.17	0.17	0.17	0.24
F-measure	A	0.25	-	-	-	-	-
	K → A	-	0.28	0.28	0.28	0.28	0.27
	K	0.34	-	-	-	-	-
	A → K	-	0.32	0.27	0.27	0.27	0.36

For the within-project predictions, our results show a mediocre value of 0.59 for the project A and a very good recall of 0.89 for the project K. The precision is very low in both cases with 0.16 for A and 0.21 for K. Hence, the F-measure is low in both projects, i.e., 0.25 for A and 0.34 for K.

The cross-projects recall predictions is lower than for within-project predictions on both A and K. The value of recall is almost always 0.48 for A, and 0.75 for K with the exception of the prediction of K when using N5 normalization, where the recall is slightly worse with 0.69. In terms of precision, we observe a mixed picture. For project A, the precision of the cross-predictions are actually slightly better than the within-project predictions with a value of 0.19 for KNN, N1, N2, and N4 and 0.18 for N5. However, the difference is rather small with 0.03 and 0.02, respectively. For Project K, the precision of the within-project prediction is better than the KNN, N1, N2, and N4 model. KNN achieves a precision of 0.20 and N1, N2, and N4 a value of 0.17. The N5 model beats the within-project prediction with a value of 0.24, i.e., a small gain of 0.03. This gain in precision seems to be the reason for the slightly worse recall. This mixture of the results is also reflected by the F-measure. For project A, the F-measure of the cross-project predictions is slightly higher with a value of 0.28 for KNN, N1,

N2, and N4 and 0.27 for N5. This gain in F-measure is due to the slightly higher precision, which offsets the lower recall. For the project K, the F-measure for KNN, N1, N2, and N4 is lower than for the within-project model. Only N5 beats the within-project prediction slightly with a value of 0.36, i.e., a very small gain of 0.02 in comparison to the within-project prediction.

In summary: the N5 cross-project prediction model performs consistently best in terms of F-measure, but has a lower recall than the within-project predictions obtained. However, the precision of all models is quite low, i.e., less than 0.25 in all cases. Accordingly to the testing experts we consulted, finding 80 % of the fault-prone instances would still be a great enhancement during the testing stages, which means that such a low precision might be acceptable for the practical implementation and is comparable to statements by Ostrand et al. [21].

3.5 Hypotheses Evaluation

In this section, we discuss the consequences of the results obtained for fault prediction on the hypotheses **H1**, **H2** as well as on our underlying research question **RQ1**.

H1: Within-Project Fault Prediction Can be Applied Successfully to Automatically Generated Code. Our results show a good recall for within-project predictions, i.e., it is possible to find the faults, but the precision is rather low. However, according to the testing experts that were involved in the Projects A and K, predicting 80 % of the fault-prone instances would still be a great enhancement during the testing stages, which means that a low precision might be acceptable in practice. Due to the high recall, we find some support for this hypothesis, but further studies need to be performed to see if the low precision really is acceptable for the practitioners.

H2: CPFP is Enabled by Auto-generated Code Within a Restrictive Setting. Our results for the cross-project prediction are not much worse than for the within-project prediction, even though we note some drop in the recall. However, the F-measure of the N5 normalization is even better than for within-project prediction. Hence, we conclude that it also depends on the practitioners point of view, if the rather low precision is acceptable.

RQ1: Can CPFP be Applied in Projects Using Automatically Generated Code and Restrictive Coding Standards Like MISRA [25] That Was Developed for the Same Target Platform? Based on our findings, we conclude that further research in this direction is warranted but the overall performance is not as good as we initially expected. Further insights are required to improve the prediction models in the future to increase precision. Moreover, a test carried out in a practical setting, where testing experts evaluate the approach in a pilot project, would be very much helpful to evaluate the question of whether or not low values of precision are really acceptable.

4 Causes for the Obtained Prediction Performance

Our results show an overall surprisingly low prediction performance, especially in terms of precision. Due to the extremely high similarity between the projects A and K in terms of the development process and, additionally, the fact that the source code is automatically generated, which should further increase the similarities, we were expecting much better prediction results. Therefore, we further investigated in the following research questions.

RQ2: What are the use fault prediction models reasons to perform low?

To answer this question we perform an in-depth analysis. We postulate three hypothesis, which we believe should be true for good fault prediction models. Then, we evaluate whether these hypothesis are true for our data and, thereby, try to gain insights into why the precision of our predictions is low. The hypotheses are the following:

H3: Metric values are strongly correlated between software projects based on automatically generated code.
H4: The available software metrics carry information about the faults.
H5: The faulty regions in the training and test data are similar.

Our rational for **H3** is that differences between metrics due to developer characteristics, etc. can be excluded. Moreover, we expect that the rules of code generators lead to repetitive patterns within the generated source code. This, in turn, should lead to patterns within the metric values, which should lead to strong correlations between the metric values, even between projects. Strong correlations between software projects are a pre-requisite for good cross-project predictions and week correlations here might by a reasons for a lower prediction performance.

With **H4**, we simply state the underlying assumption of fault prediction models based on software metrics and machine learning: that the metrics actually contain information about the location of the faults. Similarly, with **H5**, we want to investigate a general assumption for machine learning: that the distributions of the training and test data are similar. If this is not the case, the prediction model is trained using a wrong distribution that in turn decreases the prediction performance, which is somehow comparable to He et al's [13] findings.

4.1 Correlation Analysis

From the Dataset, [1], one can see a strong correlation between the static code metrics LOC, Halstead (Volume (Hv), Difficultly (Hd) and Effort (He)) and number of functions (nfunctions). A similar strong correlation has been obtained between change metrics (LOC add and removed per commit), but only a weak correlation to the author. We used Kendall's τ [16] correlation analysis to investigate the influence of the selected attributes. See Table 2 for the results. In general there is a small correlation between bugs and all 11 metrics. LOC can be

seen as the strongest correlation to bugs, but is still weak at 0.23. These results are in correlation with the attribute influence analysis in Table 3, where LOC is among the first influencing metric attributes to fault prediction.

Table 2. Kandals τ correlation analysis for metrics with bugs

		static source code attribute						change attribute				social attribute
		sloc	McCab	Hv	Hd	He	nfunctions	loc_add	loc_remove	commit_age	num_commits	author
bug	Project A	0.22	0.16	0.23	0.23	0.22	0.15	0.10	0.10	-0.12	-0.08	-0.10
	Project K	0.23	0.26	0.24	0.23	0.24	0.22	0.13	0.13	0.02	0.05	0.03

To see if the bug regions are correlated we selected all commits containing a fault and correlated their metric attributes with each other. In general there is a weak or small negative correlation between those two projects, which might be caused by different bug regions. Again the strongest correlation values are achieved by static source code metrics (LOC, Halstead, McCab).

Based on this one might conclude "within projects fault prediction" report on bad performance and "cross project fault prediction" will be impossible, at least based on the selected attribute metrics.

4.2 Information Gain

We analyzed the influence of 11 attributes using Information Gain. See Table 3 for the results, which delivers consistent values to the Kendall τ correlation analysis given in Table 2, where static source code attributes (LOC, number of functions, etc.) have got a higher correlation than change metrics (LOC_add, LOC_remove, ...). This ranking is in contrast to Graves et al. [11] or Meneely et al. [19] but similar to Curtis et al. [7]. This might be caused by the development method, as the code generator can be seen as the same author for the actual code files. Due to the fact that a code generator uses templates and the developer is limited to a finite set of blocks when designing the Matlab/Simulink Model, the code structure itself is very comparable among the methods. The influence of a developer's style is little, because every block will be generated in the same way.

4.3 Visual Analysis

In addition to the correlation analysis, we performed a visual analysis to determine possible reasons for prediction errors. To visualize the fault data, we use Principle Component Analysis (PCA) [9] to reduce the dimension of the data.

Table 3. ranked error types with explanation

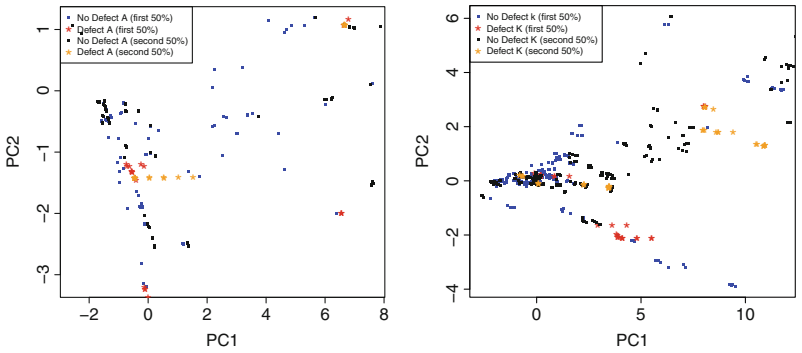
Information Gain	Rank	Attribute	Explanation
0.16637	1	Hv	Halstead Volume
0.13540	2	McCabe	McCabe Cyclomatic Complexity
0.13505	3	Hd	Halstead Difficulty
0.12628	4	He	Halstead Effort
0.12303	5	sloc	Lines Of Code
0.05445	6	nfunctions	Number of Functions
0.03483	7	commit_age	Number of Days since last commit
0.02384	8	loc_add	LOC Added since last commit
0.02358	9	loc_remove	LOC Removed since last commit
0.00948	10	num_commits	Number of total commits to file
0	11	Author	Last commiting Author

PCA is a technique that orthogonally transforms into linearly uncorrelated variables, such that the first principle component has the largest possible variance and, thereby, explains as much of the variance in the data as possible. In our case, we can explain 89 %–92 % percent of the variance within the data by using just the first two principle components. This allows us to create two-dimensional scatter plots in which we can visually compare data.

In Fig. 2(a) and (b) we show the visual representation of the data of A and K used for the within-project evaluation. The training data are the oldest 50 % revisions in the data, the test data are the newest 50 %. We have three types of interesting areas in those figures.

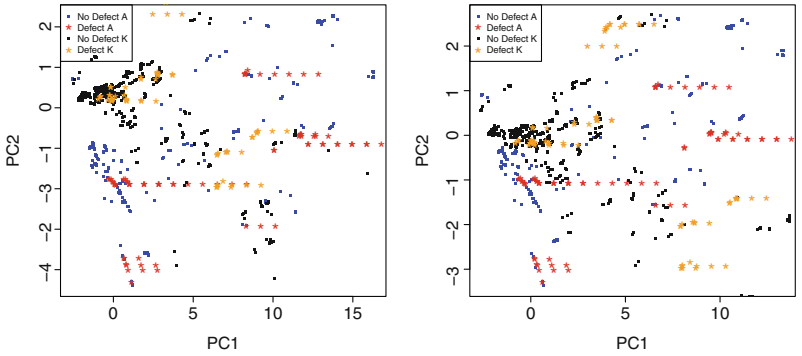
1. Defects from the training data that are the same area as in the test data. These faults should be detected correctly.
2. Defects in the training data are located in a region in which no faults are present in the test data. These faults should lead to false positives and, therefore, decrease the precision.
3. Defects in the training data are not in the same area as faults in the test data. These faults are probably not detected by the classification model and lead to a decrease in recall.

The figures show that for the most part, we are in the first areas, where we find faults in both data sets. However, in both data sets there are also areas where there is no overlap. These lead to the drops in recall and precision. As the figures also show, there are many more non-fault-prone entities than fault-prone entities. This is the reasons for the low precision: even if only a relatively small area in the training data contains faults, where there are no faults in the test data, the precision drops drastically. Moreover, in the K project, there is also the problem that the area around the coordinates (0,0) is heavily populated by both fault-prone and non-fault-prone entities in both the training and test data.



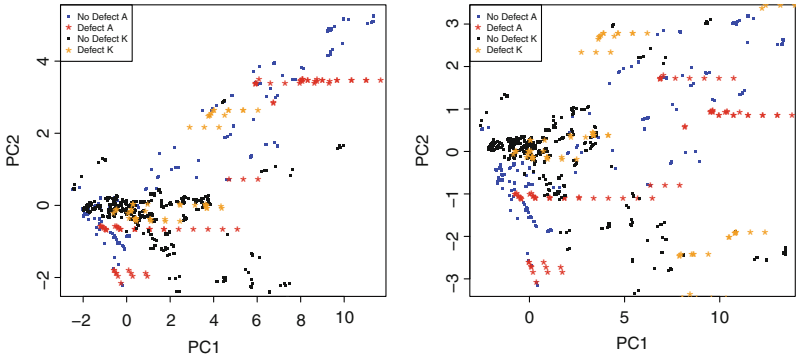
(a) PCA that visualizes the 50/50 split of A

(b) PCA that visualizes the 50/50 split of K



(c) A and K with N1 normalization applied to both

(d) A and K with N2 normalization applied to both



(e) A and K with N4 normalization applied to A

(f) A and K with N5 normalization applied to A

Fig. 2. Data visualization with PCA. The PCA is performed with all plotted data points, i.e., with the full data from A/K for the sub-figures (a) and (b) and with the data from both A and K for sub-figures (c)–(f). The first two principle components of the data explain between 89 %–92 % of the variance.

This leads to additional noise in the data and a further drop in the precision and recall. To further analyze these effects, we compare the training and test data of the cross-project predictions using the four normalization techniques N1, N2, N4 and N5 in Fig. 2(c)–(f). The areas of interest are similar.

1. Defects from the data of project A/K that are the same as in the other project. These faults should be detected correctly by the cross-project prediction.
2. Defects in the data of project A that are located in a region in which no faults are present in the project K. These faults should lead to false positives in the cross-project prediction $A \rightarrow K$ and, therefore, decrease the precision. Moreover, these faults are likely not detected in the cross-project prediction $K \rightarrow A$ and lead to a decrease in recall.
3. Defects in the data of project K that are located in a region in which no faults are present in the project A. These faults should lead to false positives in the cross-project prediction $K \rightarrow A$ and, therefore, decrease the precision. Moreover, these faults are likely not detected in the cross-project prediction $A \rightarrow K$ and lead to a decrease in recall.

As can be seen on all four plots, the faults between A and K are for the most part non-overlapping. Hence, we are for the most parts in the two "bad" areas of interest, which reduce the recall and precision. However, a closer look reveals that whereas the faulty areas themselves are not overlapping, the faults are still in somewhat similar areas in the data. For example, many faults of K are in all four plots close to the coordinates (0,0). On the Y-axis only very few instances of K are located below that cluster. Hence, fault predictors might assume the full area below those faults as fault-prone. Within that area many faulty instances of A are located, which then may be predicted correctly as fault prone. Such effects explain the still relatively good values for the recall. However, the plots also show that within the very same area below (0,0) many non-faulty instances of A are also located. They would all also be predicted as faulty, which explains the very low precision. In addition, we are interested in the general overlap between non-fault-prone instances of the two projects, i.e., how often the non-fault-prone are within the same area. The task of the normalization is exactly this: transform the non-faulty instances in such a way that they are within similar areas. We observe that the overlap depends on the normalization technique. With N1, i.e., simple min-max normalization, we see most instances of K in the top-left quadrant of the plot, whereas the instances of A are distributed scattered through the whole area of the plot. With the other normalization technique N2, N4, and N5, such a differentiation is not possible and the data seems to be more evenly distributed for both projects.

4.4 Hypothesis Evaluation

In this section, we discuss the results of the fault predictions, the hypotheses **H3**–**H5**, as well as our underlying research question **RQ2**.

H3: Metric Values Are Strongly Correlated Between Software Projects Based on Automatically Generated Code. Our correlation

analysis performance shows a very weak correlation between the two projects A and K. Therefore, we do not find support for this hypothesis. This weak correlation is the possible source for the low precision and overall bad performance of the cross-project predictions.

H4: The Available Software Metrics Carry Information About the Faults. The correlation analysis shows that most of the metrics are weakly correlated with the fault information, with the exception of the commit age, for which we find almost no correlation. Additionally, we considered the information gain of the attributes in relation to the fault information. Here, we determined that about half of the metrics carry information about the faults, however, also only weak information. The other attributes carry almost no information about the faults. If we consider this together, this is another reason for low performance, but it also explains why the predictions did not fail completely, since we found weak correlations and mutual information between the metrics and the fault information.

H5: The Faulty Regions in the Training and Test Data Are Similar. Our PCA based visual analysis shows that the faulty regions are not overlapping as expected, but only to some degree. We consider this the main source of the problems with the precision. The transfer learning only helps to fix this to a minor degree.

RQ2: What Are the Reasons for the Low Performance of the Used Fault Prediction Models? In our analysis, we found that all three hypotheses we had are actually not well supported by our data. Hence, it is not surprising that we have trouble with the overall prediction performance. The reasons we determined are merely weak instead of strong correlations of the metrics between projects, weak correlation and low mutual information of the metrics and the fault information, and a bad overlapping of the faulty regions in the training and test data.

5 Threats to Validity

We identified several threats to the validity of our results. First of all, our results are restricted to a narrow setting within the automotive industry. It is unclear how this translates to other settings, which is a threat to the external validity of our results. Moreover, the templates used by the code generators may impact our study. Different templates, which might be part of a new major release on the code generator, maybe will change our findings. Additionally, we used only two data sets for our case study. Furthermore, both data sets contain only few errors. The results may change if projects have many errors.

6 Conclusion and Outlook

In this paper, we present a case study on fault prediction models in the context of software development in the automotive industry, which is based on projects comprising automatically generated code. In our study, we considered both the

within-project and the cross-project setting. Our findings show that predicting faults is possible. However, the precision of the predictions is rather low. Due to this, we presented an in-depth analysis of possible reasons for this lack of precision. Our analysis shows that the correlation and mutual information between the software metrics and the bugs is rather weak. Moreover, the correlation between the projects is also worse then expected, as we show both with a correlation analysis as well as a visual analysis of the data.

Using these results as a starting point, we suggest multiple venues for future investigations. Since our findings regarding the metric correlation and mutual information are rather weak, we suspect that possibly using model-level metrics, instead of source code level metrics would lead to better results. Therefore, we plan to study the impact of using model-level metrics on the fault predictions. As part of this extension to model-level metrics, we also plan to investigate the influence of static code metrics, change metrics, and social metrics again. Most modern literature that considers the impact of social and change metrics is based on modern languages like Java, whereas we consider C code with restrictive coding conventions. It is unclear if the findings hold in this setting.

Acknowledgement. The authors would like to thank the project managers in charge of the analyzed projects in the data set for the long discussions and good insights they gave to development history, process and testing methods used during development and their rating of our achieved performance values on SFP.

References

1. Altinger, H., Siegl, S., Dajsuren, Y., Wotawa, F.: A novel industry grade dataset for fault prediction based on model-driven developed automotive embedded software. In: Proceedings of the 12th Working Conference on Mining Software Repositories (MSR). IEEE, Florence, Italy (2015)
2. Altinger, H., Wotawa, F., Schurius, M.: Testing methods used in the automotive industry: results from a survey. In: Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing (JAMAICA). ACM (2014)
3. Bell, R.M., Ostrand, T.J., Weyuker, E.J.: Looking for bugs in all the right places. In: Proceedings of the 2006 International Symposium on Software Testing and Analysis (ISSTA). ACM (2006)
4. Broy, M.: Challenges in automotive software engineering. In: Proceedings of the 28th International Conference on Software Engineering. ACM (2006). http://doi.acm.org/10.1145/1134285.1134292
5. Camargo Cruz, A.E., Ochimizu, K.: Towards logistic regression models for predicting fault-prone code across software projects. In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE Computer Society (2009)
6. Catal, C., Diri, B.: A systematic review of software fault prediction studies. Expert Syst. Appl. **36**(4), 7346–7354 (2009)
7. Curtis, B., Sheppard, S.B., Milliman, P.: Third time charm: Stronger prediction of programmer performance by software complexity metrics. In: Proceedings of the 4th International Conference on Software Engineering (1979)

8. Drummond, C., Holte, R.C.: C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In: Workshop on Learning from Imbalanced Datasets II (2003)
9. Karl Pearson, F.R.S.: LIII. On lines and planes of closest fit to systems of points in space. Philos. Mag. Ser. 6 **2**(11), 559–572 (1901)
10. van Gestel, T., Suykens, J., Baesens, B., Viaene, S., Vanthienen, J., Dedene, G., de Moor, B., Vandewalle, J.: Benchmarking least squares support vector machine classifiers. Mach. Learn. **54**(1), 5–32 (2004)
11. Graves, T.L., Karr, A.F., Marron, J.S., Siy, H.: Predicting fault incidence using software change history. IEEE Trans. Softw. Eng. **26**(7), 653–661 (2000)
12. He, Z., Peters, F., Menzies, T., Yang, Y.: Learning from open-source projects: an empirical study on defect prediction. In: Proceedings of the 7th International Symposium on Empirical Software Engineering and Measurement (ESEM) (2013)
13. He, Z., Shu, F., Yang, Y., Li, M., Wang, Q.: An investigation on the feasibility of cross-project defect prediction. Autom. Softw. Eng. **19**(2), 167–199 (2012). http://dx.doi.org/10.1007/s10515-011-0090-3
14. Herbold, S.: Training data selection for cross-project defect prediction. In: Proceedings of the 9th International International Conference on Predictive Models in Software Engineering (PROMISE). ACM (2013)
15. Jin-Hua, L., Qiong, L., Jing, L.: The w-model for testing software product lines. In: International Symposium on Computer Science and Computational Technology (ISCSCT) (2008)
16. Kendall, M.G.: Rank correlation methods (1948)
17. Kotsiantis, S., Kanellopoulos, D., Pintelas, P.: Data preprocessing for supervised leaning. Int. J. Comput. Sci. **1**(2), 111–117 (2006)
18. Ma, Y., Luo, G., Zeng, X., Chen, A.: Transfer learning for cross-company software defect prediction. Inf. Softw. Technol. **54**(3), 248–256 (2012)
19. Meneely, A., Williams, L., Snipes, W., Osborne, J.: Predicting failures with developer networks and social network analysis. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE) (2008)
20. Nam, J., Pan, S.J., Kim, S.: Transfer defect learning. In: Proceedings of the 35th International International Conference on Software Engineering (ICSE) (2013)
21. Ostrand, T.J., Weyuker, E.J.: The distribution of faults in a large industrial software system. ACM SIGSOFT Softw. Eng. Notes **27**, 55–64 (2002)
22. Ostrand, T.J., Weyuker, E.J., Bell, R.M.: Predicting the location and number of faults in large software systems. IEEE Trans. Softw. Eng. **31**(4), 340–355 (2005)
23. Rana, R., Staron, M., Hansson, J., Nilsson, M.: Defect prediction over software life cycle in automotive domain. In: Proceedings of the Joint International Conference on Software Technologies (ICSOFT) (2014)
24. Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press, Cambridge (2002)
25. The Motor Industry Software Reliability Association: MISRA-C:2004 - Guidelines for the use of the C language in critical systems, 2nd edn. MISRA, Warwickshire (2004)
26. Turhan, B., Menzies, T., Bener, A., Di Stefano, J.: On the relative value of cross-company and within-company data for defect prediction. Empirical Softw. Eng. **14**, 540–578 (2009)
27. Watanabe, S., Kaiya, H., Kaijiri, K.: Adapting a fault prediction model to allow inter language reuse. In: Proceedings of the 4th International International Workshop on Predictor Models in Software Engineering (PROMISE). ACM (2008)

28. Zhang, F., Mockus, A., Keivanloo, I., Zou, Y.: Towards building a universal defect prediction model. In: Proceedings of the 11th Working Conference on Mining Software Repositories (MSR) (2014)
29. Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B.: Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the the 7th Joint Meeting European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE) (2009)