A Novel Industry Grade Dataset for Fault Prediction based on Model-Driven Developed Automotive Embedded Software

Harald Altinger*, Sebastian Siegl*, Yanja Dajsuren‡ and Franz Wotawa§
*Audi Electronics Venture GmbH, 85080 Gaimersheim, Germany
Email: [firstname].[lastname]@audi.de
‡Eindhoven University of Technology, Eindhoven, The Netherlands
Email: y.dajsuren@tue.nl
§Institute for Software Technology, Graz University of Technology, 8010 Graz, Austria
Email: wotawa@ist.tugraz.at

***Abstract*—In this paper, we present a novel industry dataset on static software and change metrics for Matlab/Simulink models and their corresponding auto-generated C source code. The data set comprises data of three automotive projects developed and tested accordingly to industry standards and restrictive software development guidelines. We present some background information of the projects, the development process and the issue tracking as well as the creation steps of the dataset and the used tools during development. A specific highlight of the dataset is a low measurement error on change metrics because of the used issue tracking and commit policies.**

I. Introduction

In this paper, we present a dataset acquired from three different automotive embedded software projects that have been developed in-house at Audi Electronics Venture GmbH (AEV). The dataset comprises three tables for every project and is available for download at [1]. Two of these datasets are from projects, in which safety functionalities has to be ensured during development. The quality of the datasets is hence very high, as restrictive issue tracking policies were applied and assured during carrying out the development process. All three programs were implemented making use of model-driven approaches and realized using Matlab/Simulink accordingly to model guidelines like the one provided by the MathWorks Automotive Advisory Board [2] (MAAB). The source code was automatically generated from the Matlab/Simulink models. There were no manual code changes within the source files. All code parts conform to Motor Industry Software Reliability Association (MISRA) C 2004 [3] and represent a high level of code quality. As examined by Stamelos et al. [4], not all Open Source projects conform to the software industries high quality standards. Hence, we contribute a dataset that fulfils industry grade code to be publicly available for research and further investigations in data mining applied to software repositories.

The available dataset comprises static model metrics (Modularity, Size, Complexity, etc.), static source code metrics (Lines Of Code (LOC), McCabe [5], Halstead [6]), and change metrics (number of commits, number of bugs, authors, etc.). The code has been managed using a version control system with an integrated issue tracking system. Due to restrictive commit policies we are able to exactly determine the files that have to be changed in order to fix a bug. Due to confidentially reasons, we are not allowed to publish the source code nor the model. Instead we publish all information that is necessary for investigating on deriving fault prediction models from metrics data is available. In addition we give a detailed description of how we created the dataset in the rest of this paper.

The paper is structured as follows: the software development process and tools are described in Section II, an overview of the projects is provided in Section III, the steps to create the dataset are illustrated in Section IV. The structure of the dataset and its field description are introduced in Section V, followed by the final conclusions and findings on the dataset in Section VII.

II. Development Work-flow and Tools

Automotive software development follows the W-Development process, c.p. Jin-Hua et al. [7], with requirements, a model design, code generation, etc. The left side of the W is the specification, in the center are various testing preparations, and on the right side of the W are the test and analyse steps.

Software development starts with the aggregation of system requirements and interface definitions along with the software architecture. Using these specification documents the function developer starts to design the models. During development there are two important milestones, *feature freeze* and *100% software*. After *feature freeze*, no new functionality is allowed to be added anymore. After *100% software*, the development itself is finished. Between these milestones and Start Of Production (SOP), the automotive equivalent to a release date of the software with a new car model, there will be the testing phase and bug fix commits, but no feature enhancements.

In a recent survey Altinger et al. [8] presented tools and methods used within the development of automotive software. IBM Doors is the most common tool to specify requirements and Matlab/Simulink as the most wide-spread modelling tool

978-0-7695-5594-2/15 $31.00 © 2015 IEEE
DOI 10.1109/MSR.2015.72

for embedded automotive functionality. These two tools were also in use during the whole development in the three projects from which we obtained the dataset.

To generate code all three projects used dSpace TargetLink, [9], with a fixed set of settings after an initial optimization phase. The Matlab/Simulink models were created using blocks from libraries, which contained block sets for generating code with TargetLink. Using this block set, a model simulation (Model in the Loop (MiL)) can be done in Simulink, as well as a software simulation (Software in the Loop (SiL)) of the auto generated target code. In most cases code has been generated prior to a periodic software release, which has been less frequent than changes in the models.

The testing activities comprise mainly 4 platforms and integration levels, i.e. MiL, SiL, Hardware in the Loop (HiL) and car tests (cp. Bringmann et al. [10] for a detailed explanation). Test cases were designed based on the requirements for each project and executed as regression tests after each release.

As a version control system PTC Integrity [11] was used. This tool offers features comparable to Subversion (commit, checkout, branch, tag, etc.). In addition, it contains an integrated ticket and issue tracking system and can be controlled via policies. To be able to commit every contributor must specify a ticket and classify his or her code changes. Tickets can be created by a member of the development team, a test engineer or the project manager. There are main ticket categories, e.g. *change request* to commit new features and interface adoptions or *trouble reports* if undesired behaviour of the implementation has been detected. Among other things, trouble reports contain sub categories: 'real code error', 'specification error', etc. The ticket categories can be adopted if required. Every code commit to a ticket is organized as so called 'change package', which enables us to track which files have been changed, e.g. to fix a bug.

Bugs can be reported during all the testing and the development stages.

III. Project description

Due to confidentially reasons we do not name the projects or list the car and the model year when they were released. In most cases, automotive software runs on Electronic Control Unit (ECU), an embedded hardware with limited CPU and memory resources. Depending on target functionality of the ECU it contains I/O, Sensors and at least one bus interface (e.g. Controller Area Network (CAN)). If a ECU operates with AUTomotive Open System ARchitecture, [12] (AUTOSAR), there is an Run-Time Environment (RTE) for the specific hardware to manage I/O, tasks, etc. All functional software is organized as Software Component (SWC), similar to tasks executed on a computer. We report on three of such SWCs, see Table I. Project A is a novel interior comfort functionality, Project L is a embedded body functionality with complex logic, Project K is a embedded body functionality. For Project K and Project L the process-related standards were high and well assured. As a consequence, the change management and error tracking followed a even more strict rules, which allow for a reliable data analysis basis

Note that we report only the number of test cases on MiL and SiL because they specific target the functionality of the SWC. Test cases on HiL and the whole car level target overall functionality where multiple SWC are required. Therefore, one single test case can not be traced back to functions of a single SWC.

IV. Dataset creation

To create the dataset we used a collection of python scripts. The first task has been to extract the models and source code from the PTC Integrity repository and commit them into a temporary Subversion (SVN) repository without affecting the version history and maintaining a link between Integrity version numbering and SVN revisions. After this we queried the issue tracking system for tickets marked as 'trouble ticket' with subject category 'real code bug'. The linked change packages gave us the affected files and revision of a code fix. We implemented Kim et al. SZZ-Algorithm [13] Step 1 to 4 using ScooterSoftware's 'Beyond Compare' [14] to diff the files. The command 'svn blame' could prevail the version number when the source code line has been introduced.

The code generator is parametrized to report all generated source code files per model file and every model file contains an entry *LastModifiedDate* as every generated source code file *Generation date*. This enabled us to link the source code to the model revision.

The static source code measurement has been performed using 'LocMetric' [15] and 'CMetric' [16]. The model metrics have been measured with the 'assaqalw' tool [17], which is an extension of the modularity metrics tool, c.p. Dajsuren et al. [18]. It is a Java tool that reads Simulink MDL files with the standard structural format and generates a metrics file with the list of subsystems and the respective model metrics. The exact calculation equations are listed within the tool's documentation.

The md5 hash has been calculated with python's `hashlib`.

V. Description of Data

The dataset contains three tables per project representing the measured values. Either a table contains a md5 hash as unique key or a combination of 'filename' and 'revision' fields.

Static model metrics, e.g. Table II, were acquired with the 'assaqalw' tool, which was described in the section IV and further illustrated in Dajsuren [17]. Basically a Matlab/Simulink file can be saved as a 'MDL' file, which is an ASCII file containing information describing the model. Simulink models may contain multiple sub-modules that will be reported in separate rows. The field 'last_modified_date' has been extracted from the LastModifiedDate entry within the file. Table II lists the tools used for every field.

Static source code metrics, e.g. Table III, were acquired with LocMetric, [15], and CMetric, [16], file by file. The field 'last_modified_date' has been extracted from the TargetLink

TABLE I

PROJECT DESCRIPTION AND DATA OVERVIEW: 'NUMBER OF FILES' REPRESENTS THE UNIQUE NUMBER OF FILES, 'COMMITTED FILES' REPRESENTS THE SUM OF ALL COMMITS OVER ALL FILES, 'ERROR PRONE FILES' REPRESENTS THE NUMBER OF FILES MARKED AS 'REAL CODE BUG' OVER ALL COMMITS

	num. Requirements	num. Sub-projects	LOC	num. Testcases	num. Authors	num. src. files	num. mdl. files	committed files	error prone files	Software Type	AUTOSAR	Safety functionality
Project A	304	13	12,465	185	4	45	26	1,782	78	logic, timing dependent behaviour	yes	no
Project L	600	8	10,113	680	3	20	47	2,892	76	logic, timing dependent behaviour	yes	yes
Project K	900	24	36,526	695	5	53	48	2,481	329	mainly logic operations and branching	yes	yes

generated timestamps Table III lists the tools used for every field.

Change metrics and bug statistic, e.g. Table IV, represent the data from the issue tracking system. The date field represents the last update to the change set, 'svnrev' the SVN revision of the affected file, whereas 'project_rev' is the original revision from the PTC Integrity system. The field 'num_bugs_trace' represents the accumulated number of faulty LOC from the diff process described within section IV. The files commit message has been analysed upon the words 'bug' and 'error'. Their number is reported within the field 'pot_bugs'. If the commit has been a bug fix the field 'num_bugs' will contain 1, and -1 otherwise.

TABLE II

TABLE CONTAINING MODEL METRICS

fieldname	comment	tool
filename	files name	python
hash	md5 file hash	python
project_name	subprojects name	[17]
Subsystem	Matlab subsystem	[17]
SubsystemPath	subsystem path	[17]
project_rev	file revision [major.minor]	python
MOD_NOB	Number of Blocks	[17]
MOD_DSC	Degree of Subsystem Coupling	[17]
MOD_DoS	Depth of a Subsystem	[17]
MOD_NCS	Number of Contained subsystems	[17]
MOD_SZ	Subsystem size	[17]
ANZ_hCMX_n	number of distinct Simulink block types and signals	[17]
ANZ_hCMX_N	number of Simulink blocks and signals	[17]
ANZ_hCMX_V	Halstead Volume metric	[17]
ANZ_hCMX_D	Halstead Difficulty metric	[17]
ANZ_hCMX_E	Halstead Effort metric	[17]
ANZ_mCMX	McCabbe Cyclomatic complexity	[17]
n	number of blocks	[17]
gini	gini index	[17]
csu	component size uniformity	[17]
ANZ_SB	Subsystem Balance	[17]
Sig_gini	Signal-based gini index	[17]
Sig_csu	Signal-based component size uniformity	[17]
ANZ_SSB	Subsystem Signal Balance (based on signal density)	[17]
UND_NrAnnot	Number of Annotations	[17]
last_modified_date	last save timestamp	python

TABLE III

TABLE CONTAINING SOURCE CODE METRICS

fieldname	comment	tool
filename	filename	python
project_name	subproject name	python
hash	md5 file hash	python
lm_LOC	LOC	[15]
lm_SLOCP	Physical Executable LOC	[15]
lm_SLOCL	Logical Executable LOC	[15]
lm_MVG	McCabe VG Complexity	[16]
lm_BLOC	Blank LOC	[15]
lm_CNSLOC	Code and Comment LOC	[15]
lm_CLOC	Comment Only LOC	[15]
lm_CWORD	Commentary Word	[15]
lm_HCLOC	Header Comment LOC	[15]
lm_HCWORD	Header Commentary Words	[15]
h_N1	number of total operators	[16]
h_N2	number of total operands	[16]
h_n_1	number of unique operators	[16]
h_n_2	number of unique operands	[16]
h_V	Halsted volumen	[16]
h_D	Halsted difficulty	[16]
h_E	Halsted effort	[16]
last_modified_date	TargetLink generation time	python
src_mdl_file_hashes	hash from mdl	python

TABLE IV

TABLE CONTAINING CHANGE METRICS

fieldname	comment
filename	filename
project_name o	subproject name
author	authors name [surename.firstname]
filetype	type of file (header or source)
project_rev	PTC Integrity revision number
date	date of entry
svnrev	SVN revision number
num_bugs_trace	number of buggy lines
pot_bugs	number of 'bug', 'error' in commit message
num_bugs	number of fixed bugs

VI. FIRST DATA ANALYSIS

In this section, we briefly discuss the first results obtained when analysing the described dataset. We used Kendalls τ to determine correlation between LOC and McCabe/Halstead, which where similar to the findings discussed by Curtis et al. [19]. Analysing all files over all revisions one is

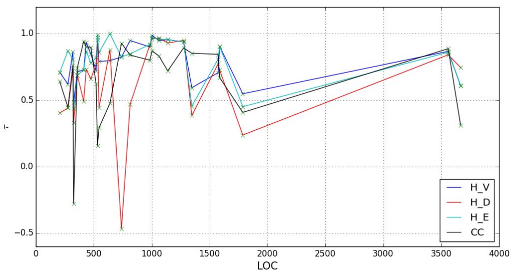

Fig. 1. Kendalls τ analysis between LOC and McCabe/Halstead for project K over all revisions and files

able to see more strong correlations than weak correlations between LOC and McCabe/Halstead (see Figure 1).

Based on the initial analysis of the static model metric values project A subsystems are bigger ($MOD\_NOB = 933$) than project L subsystems ($MOD\_NOB = 363$). In the preliminary evaluation of modularity metrics, it was identified that the higher the hierarchical level (higher MOD_DoS) more difficult to understand the system is, c.p. Dajsuren et al. [18]. The complexity values of these projects also indicate that the project K is the most complex (e.g. $max(ANZ\_hCMX\_D) = 8.02$, $max(ANZ\_mCMX) = 54$), project A ($max(ANZ\_hCMX\_D) = 8.02$, $max(ANZ\_mCMX) = 18$) and project L ($max(ANZ\_hCMX\_D) = 5.5$, $max(ANZ\_mCMX_) = 14$). The subsystem balance metrics (ANZ_SB and ANZ_SSB) indicate that all three projects contain similar amount of balanced subsystems (well decomposed systems). Higher values of subsystem balance metric values better the subsystem decomposition, c.p. Bouwers et al. [20], therefore we compared the number of subsystems with the value of 1. However, there will be further evaluation on defining the thresholds of the model metrics values.

VII. Conclusion and Outlook

In this paper we discussed a dataset obtained from real projects carried out in the automotive industry based on standard software development processes. Although, the source code of the projects cannot be published because of confidentially reasons, all information necessary to investigate on the relationship between metrics and faults is available in the dataset. The data provided was anonymized. In addition we reported on first analysis results. The obtained strong correlation characteristics may be due to the structure of automatically generated code. Further investigations are needed to analyse the finding. There are hardly any studies on this topic in this domain to our best knowledge. It would be very much interesting to apply available fault prediction methods on the presented dataset, due to the high quality of the bug information given in the set. In a recent study Bell et al. [21] state that change metrics outperform static metrics when applied to source code. This dataset delivers novel static metrics on models, which might give new and different insights on fault prediction. Due to the restrictive commit policies and the safety related nature of the projects the quality of the data is very high, in particular the bug report data.

Acknowledgement

The authors thank the project managers for their patience during interviews. The third authors is funded under grant agreement nr. 289011 (OPENCOSS)

References

[1] H. Altinger, "Dataset on automotive software repository," Feb. 2015. [Online]. Available: http://www.ist.tugraz.at/_attach/Publish/AltingerHarald/MSR_2015_dataset_automotive.zip

[2] T. Mathworks, "Mathworks automotive advisory board checks (MAAB)," 2014. [Online]. Available: http://de.mathworks.com/help/slvnv/ref/mathworks-automotive-advisory-board-checks.html

[3] The Motor Industry Software Reliability Asso- and ciation, *MISRA-C:2004 - Guidelines for the use of the C language in critical systems*, 2nd ed. Warwickshire: MISRA, 2004.

[4] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code quality analysis in open source software development," *Information Systems Journal*, vol. 12, no. 1, p. 4360, 2002.

[5] T. J. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, no. 4, p. 308320, 1976.

[6] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA: Elsevier Science Inc., 1977.

[7] L. Jin-Hua, L. Qiong, and L. Jing, "The w-model for testing software product lines," in *Computer Science and Computational Technology, 2008. ISCSCT'08. International Symposium on*, vol. 1, 2008, p. 690693.

[8] H. Altinger, F. Wotawa, and M. Schurius, "Testing methods used in the automotive industry: Results from a survey," in *Proc.\ JAMAICA*. San Jose, CA: ACM, Jul. 2014, p. 16.

[9] dSpace, "TargetLink," 2015. [Online]. Available: https://www.dspace.com/de/gmb/home/products/sw/pcgs/targetli.cfm

[10] E. Bringmann and A. Kramer, "Model-based testing of automotive systems," in *Software Testing, Verification, and Validation, 2008 1st International Conference on*, 2008, pp. 485–493, IEEE.

[11] PTC, "Integrity," 2015. [Online]. Available: http://www.ptc.com/product/integrity/automotive

[12] C. Autosar, "Autosar," Mar. 2014. [Online]. Available: www.autosar.org

[13] S. Kim, T. Zimmermann, K. Pan, and E. J. Whitehead, "Automatic identification of bug-introducing changes," in *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*, 2006, p. 8190.

[14] S. Software, "Beyond compare 4," 2015. [Online]. Available: http://www.scootersoftware.com/

[15] locmetrics.com, "LocMetric," 2015. [Online]. Available: http://www.locmetrics.com/

[16] H. Israel, "CMetric," 2015. [Online]. Available: https://github.com/MetricsGrimoire/CMetrics

[17] Y. Dajsuren, "On the design of architecture framework and quality evaluation for automotive systems," PhD Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, May 2015.

[18] Y. Dajsuren, M. G. van den Brand, A. Serebrenik, and S. Roubtsov, "Simulink models are also software: Modularity assessment," in *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*. ACM, 2013, p. 99106.

[19] B. Curtis, S. B. Sheppard, and P. Milliman, "Third time charm: Stronger prediction of programmer performance by software complexity metrics," in *Proceedings of the 4th international conference on Software engineering*, 1979, p. 356360.

[20] E. Bouwers, J. P. Correia, A. van Deursen, and J. Visser, "Quantifying the analyzability of software architectures," in *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*, 2011, p. 8392.

[21] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Looking for bugs in all the right places," in *Proceedings of the 2006 international symposium on Software testing and analysis*, 2006, p. 6172.