Influence of Software Complexity on ISO/IEC 26262 Software Verification Requirements

Miroslaw Staron*, Rakesh Rana*, Jörgen Hansson†
*Computer Science and Engineering
Chalmers / University of Gothenburg
Email: name.surname@gu.se
†University of Skövde
Email: jorgen.hansson@his.se

***Abstract*—With the emergence of new IT technologies our vehicles evolve from being machines to becoming self-driving cyber-physical systems of systems. The abilities of modern computers and software allow the car manufacturers to develop and deploy increasingly complex functions such as automated parking, collision avoidance or the upcoming self-driving. However, as the new functions are implemented, the electronics and software of the cars has more possibilities to intervene with the driver's actions, which leads to the more careful need to evaluate the decisions made by software. In this talk we explore how the growing complexity of software requires even more effort to validate it in the context of ISO/IEC 26262. Our results show that soon we need to change the way we work with verification and validation as the growing complexity makes it virtually impossible to achieve full certainty that the software is correct.**

I. Introduction

Modern vehicles have an increased amount of software which is used to control functionality which is safety critical [1]. Since the software-based functions become more advanced and can intervene with the driver's decisions it is important that they fulfill the safety requirements. ISO/IEC 26262 standard for functional safety in the automotive domain is one of the major baselines in this area, the other ones are the practices of the automotive software development companies (both vehicle manufacturers and their suppliers).

In this talk we discuss the challenges related to the growing complexity of software and its impact on the verification requirements in ISO/IEC 26262. We address the following research questions:

1) What is the complexity of software in modern vehicles?
2) Which ISO/IEC 26262 verification methods are influenced by software complexity?
3) Which non ISO/IEC 26262 required verification methods exist that could increase the effectiveness of verification and decrease its cost?

In this paper we base our work on the data set presented in [2] from the automotive domain which contains the data about complexity, size and defects found in software components. The data set has been published recently and can be used to draw conclusions about the status of software in modern cars. To address the second research question we conduct an analysis of testing methods mentioned in ISO/IEC 26262 or which could be used to fulfill the requirements posed by the standard (e.g. branch coverage).

In order to address the third research question we perform a semi-systematic literature review and a meta-analysis of publications in the area of fault injection and reliability prediction.

II. Software complexity

In general software complexity can be measured in multiple ways, but there is a small number of measures which have been found to be correlated with each other – e.g. McCabe cyclomatic complexity, lines-of-code. The inherent correlations (c.f. [3]) allow us to simplify the problem to only one of them (for the sake of the discussion) – we choose the McCabe complexity due to its spread in practice. In short the metric measures the number of independent execution paths in the source code.

In the automotive sector, in the data from open domain we find that the complexity of software modules is highly over the theoretical limits of 30 (execution paths) as it is presented in figure 1.

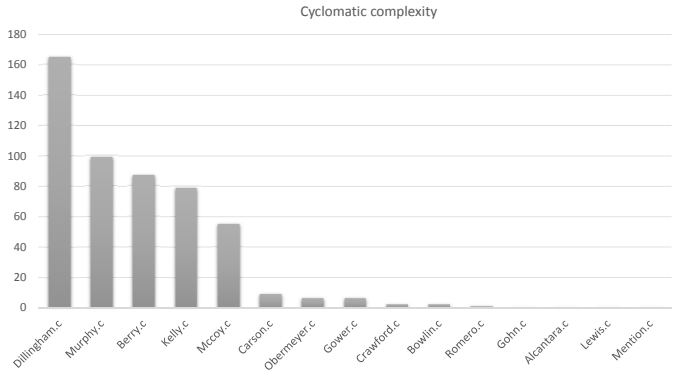

Fig. 1. Complexity of software modules (C programming language) as a McCabe cyclomatic complexity

What the data shows is that there are components where the number of execution paths is over 160 which means that only to test each of the execution path once there is a need for 160+ test cases. However in order to achieve full coverage one needs more than 500 test cases for the entire component. If we need to test each path as a positive and negative case (so called boundary case) we need to at least double the number

of test cases. Exploring the other metrics provided in the same data set shows that the trends are very similar – the numbers are highly over the theoretical complexity limits.

These numbers indicate that it is increasingly more difficult to provide full verification of the software functionality in order to ensure safety of software systems. Therefore we need new approaches than just testing.

III. ISO/IEC 26262 verfication requirements for software

Chapter 6 of the ISO/IEC 26262 standard explicitly recognizes the problems related to software complexity, but does not provide normative guidelines for what the levels should be ("Enforcements of low complexity" by modelling and coding guidelines). Such a situation is understandable as there is no single measure of complexity or a consensus on what complexity really is.

However, in general there is a consensus that the complexity is in correlation to size of software (c.f. [3], [4]). So in this talk we discuss the measures of both complexity and software size – e.g. McCabe cyclomatic complexity for source code and number of blocks/signals for Simulink models.

In another part of ISO/IEC 26262 another types of software complexity are recognized such as coupling ("Restricted coupling between software components"), which is related to the measures of complexity by Henry and Kafura which we have explored in our previous studies on the architecture level of an electrical system of a car [5].

In our talk we present the principles of how to calculate these types of complexity measures and what that means for the testing/verification methods prescribed in ISO/IEC 26262, e.g. control flow monitoring at the architectural level, and interface tests at the unit of code level. We show the difficulty of achieving high levels of coverage based on examples and reflect on the difficulty of achieving certainty of code quality based on the data set from the public domain from the embedded software in cars [2].

IV. Alternative methods

Mutation testing is one of the techniques where the object of analysis is not only the system itself (SUT – System Under Test), but also the test suite. In mutation testing parts of correct source code are replaced by a defective code – e.g. by changing the condition of an if statement from "true" to "false". When testing the replaced code (so called the mutant) the defect should be found. Using this kind of testing can improve the "certainty" in the results of the verification process as described by Rana et al. [6], [7].

Another way of achieving higher reliability of software is to perform defect-oriented root cause analyses based on defect classifications (c.f. [8], [9]) and using that in improving predictions (c.f. [10]). We also explore how the use of abstractions and transformations can increase the predictability of software development processes (c.f. [11]).

V. Conclusion

In this talk we highlight the challenges with the growing complexity of software in modern cars. We present the trends in the complexity growth and we discuss the impact of these trends on the transportation systems in general – e.g. autonomous driving, safety assessment. We present examples of the challenges based on the ISO/IEC 26262 standard and a publicly available data set from the automotive domain.

We contribute to the discussion on the implications of software-controlled functionality both for the vehicles and for the entire transportation network. We conclude the talk by forecasting a possible development of software complexity based on extrapolating the trends in the open data set and based on the new features which the automakers present today in their marketing materials.

Acknowledgment

The research presented here is done under the VISEE project which is funded by Vinnova and Volvo Cars jointly under the FFI programme (VISEE, Project No: DIARIENR: 2011-04438).

The research presented here is done under the SSF mobility project (Swedish Strategic Research Foundation) under the grant number SM13-0007.

References

[1] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 33–42.

[2] H. Altinger, S. Siegl, Dajsuren, Yanja, and F. Wotawa, "A novel industry grade dataset for fault prediction based on model-driven developed automotive embedded software," in *12th Working Conference on Mining Software Repositories (MSR)*. MSR 2015, 2015.

[3] V. Antinyan, M. Staron, J. Hansson, W. Meding, P. Osterström, and A. Henriksson, "Monitoring evolution of code complexity and magnitude of changes," *Acta Cybernetica*, vol. 21, no. 3, pp. 367–382, 2014.

[4] D. I. Sjøberg, B. Anda, and A. Mockus, "Questioning software maintenance metrics: a comparative case study," in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2012, pp. 107–110.

[5] D. Durisic, M. Nilsson, M. Staron, and J. Hansson, "Measuring the impact of changes to the complexity and coupling properties of automotive software systems," *Journal of Systems and Software*, vol. 86, no. 5, pp. 1275–1293, 2013.

[6] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Increasing efficiency of iso 26262 verification and validation by combining fault injection and mutation testing with model based development." in *ICSOFT*, 2013, pp. 251–257.

[7] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Torner, "Evaluating long-term predictive power of standard reliability growth models on automotive systems," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*. IEEE, 2013, pp. 228–237.

[8] N. Mellegård, M. Staron, and F. Törner, "A light-weight software defect classification scheme for embedded automotive software and its initial evaluation," *Proceedings of the ISSRE 2012*, 2012.

[9] N. Mellegård and M. Staron, "Characterizing model usage in embedded software engineering: a case study," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. ACM, 2010, pp. 245–252.

[10] M. Staron and W. Meding, "Predicting short-term defect inflow in large software projects–an initial evaluation," *11th International Conference on Evaluation and Assessment in Software Engineering, EASE*, 2007.

[11] L. Kuzniarz and M. Staron, "On practical usage of stereotypes in uml-based software development," *the Proceedings of Forum on Design and Specification Languages, Marseille*, 2002.