ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 262-266
doi: 10.18178/wcse.2017.06.045

Feature-Testing of Engine Electronic Control Unit Software

Sven Dominka [1+], Michael Dübner [1], Dominik Ertl [1], Michael Mandl [1]

[1] Robert Bosch AG, Austria

Abstract. Modern engine electronic control unit software contains a huge amount of features and the number is continuously increasing. This leads to the risk of undesired mutual reactions between features, so called feature interactions. Hence finding those feature interactions in an efficient way is an important task. This paper presents new approaches for efficiently testing those features: Feature-Test-Levels and Advanced Feature Testing. Feature-Test-Levels are especially useful for non-feature-oriented SW structures.

Keywords: feature interaction, automotive, software, testing

1. Introduction

Due to increasing software (SW) complexity, among others due to increasing number of features, testing of software becomes more and more time-consuming ([1]). Hence, countermeasures are needed.

1.1. Features Within Engine Electronic Control Unit

Because of increasing demands regarding efficiency, performance, comfort, safety and pollution reduction, more and more features are included in a combustion engine ([2]). Features can have mutual effects on each other ([3]). Those effects are either desired or undesired. Automotive features are different to pure-software features as they are part of a cyber-physical system where software, hardware and the environment strongly interact with each other ([4]). In contrast to other domains, in which feature interaction typically occurs directly within the software ([5],[6]), in case of combustion engines, reciprocal effects often appear on a physical level. One example is the air mass within the combustion chamber, which is influenced by a multitude of engine actuators and hence by different features.

Examples of engine features are:

- Catalyst Heating
- Cruise Control & Speed Limiter
- Hybrid Operation
- Start/Stop Operation & Coasting
- Half Engine Mode
- Stratified Operation Mode

The software structure of engine electronic control units (ECU) can be either feature-oriented or non-feature-oriented. Non-feature-oriented engine ECU software is often based on engine structure and engine physics, which allows better mapping of software to hardware and generally supports better understanding, modification and extension of the engine base functionality. Figure 1 shows an example of a fairly simplified non-feature-oriented structure of an Engine ECU application software. The application software covers, among others, the coordination of the various torque requesters, the calculation of the required air mass, fuel mass and igniting timing and the inclusion of various sensor values, like fuel system pressures, intake manifold pressures, exhaust system oxygen ratio. In this type of SW structure, features are often

+ Corresponding author.
E-mail address: *sven.dominka@at.bosch.com*

implemented in several modules, which are often part of different subsystems, see Figure 2. Some features are actually distributed over most parts of the application software.

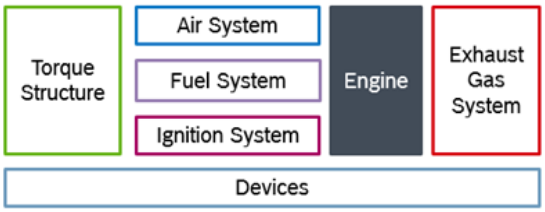

Fig. 1: Example of a Simplified Structure of an Engine ECU Application Software

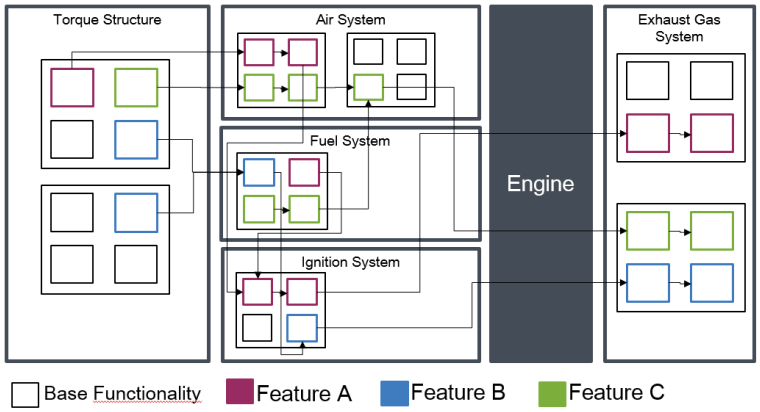

Fig. 2: Features within a non-feature-oriented Application Software Structure

1.2. Challenge in Testing of Features

Although, modern combustion engines and the respective Engine ECU software contains many features, it is typically only a subset of the existing ones. Which features are selected by OEMs might depend on various factors, like target market, costs and brand-specific decisions. Hence, suppliers must cope with a huge amount of different feature configurations that are included within Engine ECU software. The multitude of features in a non-feature oriented software structure combined with the various feature configurations poses significant challenges in testing of features.

- Typically, tests can be divided into 4 levels: unit, integration, system and acceptance testing ([7]). Unit and integration test levels are often not beneficial for validating features. A module or unit typically includes functionality of several features, but does rarely include a complete feature. The same is often true for grouped modules, like subsystems (integration level). Within the system test level boundaries, all features are contained. However, due to the lack of explicit feature boundaries, a black box validation against requirements is rather difficult.
- Due to the concatenated nature of engine ECU features, combining various features can lead to undesired feature interaction. Changes within a single feature can lead to misbehaviour in an indefinite number of variants. Hence, after changing or adding features, various feature combinations must be intensively tested in respect to undesired feature interaction.

1.3. Approaches for Efficient Feature-Testing

To address the above mentioned challenges in testing of features, we investigated the following two approaches for efficient feature testing:

1) Specifying of Feature-Test-Levels
2) Advanced Feature Testing

In [4], the terms feature and interaction-testing are defined as follows: “Feature testing is the process of validating feature requirements by testing whereas interaction testing is the process of validating integration of features to compose an application as well as their possible ways of interaction with other features”. For

the scope of application that is described in this paper, above definition is complemented by the term ‘verification’.

2. Feature-Test-Level

The goal of the introduction of feature-test-levels is the, at least notional, clustering of de-clustered features in a non-feature-oriented SW structure. For validating the software against requirements, black-box testing is typically applied. This however, requires explicit system boundaries and interfaces. For testing of features, such explicit boundaries and interfaces are needed. As in a non-feature-oriented SW structure, such boundaries hardly exist, they have to be created in the context of feature-test-levels. This is solely done on a conceptual base. No software needs to be changed. To achieve this goal, the base principles of software engineering, like abstraction, information hiding, modularity and separation of concerns are a great help.

The first step here is to identify feature specific functionality and general functionality that is required by the feature to fully work (but not specific to a single feature). Second step is the abstraction by ignoring function parts which are out of focus and by removing unit borders. What remains is the main path of the feature variables, representing the data and control flow of the specific feature. Here it is distinguished between the inner feature and the outer feature. The inner feature merely contains specific feature functionality that is not used by any other software parts. The outer feature embraces the inner feature and additionally contains function parts that are needed for the feature to fully work. Those function parts might be assigned to base functionality and at the same time be part of several other outer features. For both, inner and outer features, an explicit interface should be specified.

Figure 3 shows an example of the feature “Cruise Control” and its widespread distribution over the application software of the Engine ECU. Figure 4 shows the re-clustered version of “Cruise Control”.

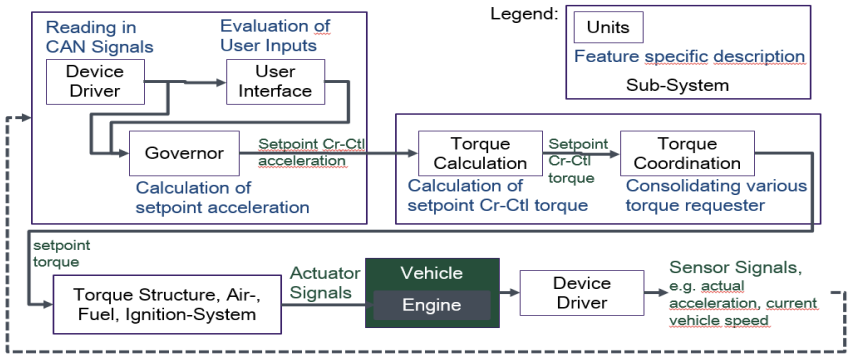

Fig. 3: “Cruise Control” Feature within Application Software

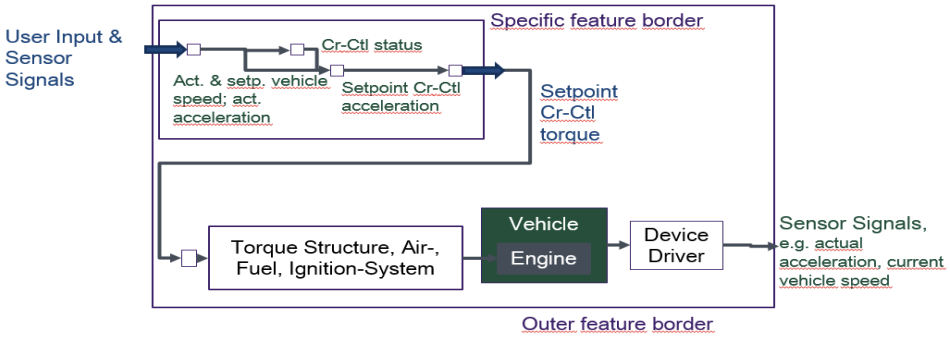

Fig. 4: Re-clustering of “Cruise Control” Feature

The created feature design specification is the base for test case specification. Depending on the main purpose for test case execution, the test case can focus on inner feature or outer feature borders. Test specifications based on the inner feature border blend out any possible interplay with the remaining system. This allows a focussed verification and validation of the requirement specification of the specific feature. Especially for error diagnostics, the inner feature borders are helpful to prove the correctness of the specific

software parts. Test specification based on the outer feature borders, on the other hand, allows validation on system level, as many feature requirements specify its desired behaviour with system interfaces.

Especially for error diagnostics, the inner feature borders are helpful to prove the correctness of the specific software parts. Test specification based on the outer feature borders, on the other hand, allows validation on system level, as many feature requirements specify its desired behaviour with system interfaces.

Figure 5 shows an example of two simple test cases for "Adaptive Cruise Control", one based on inner feature border and the second based on outer feature border. In both cases, the vehicle acceleration is evaluated based on vehicle speeds and accelerator pedal state. The only difference between the two test cases lies in the output interface that is used for evaluation. In case of the inner feature border, the desired (setpoint) acceleration is evaluated. In case of the outer feature border, the actual acceleration is checked. By using the actual acceleration (that is outer feature border) it can be validated, if the vehicle (and not only the specific feature software part) is showing the desired behaviour.

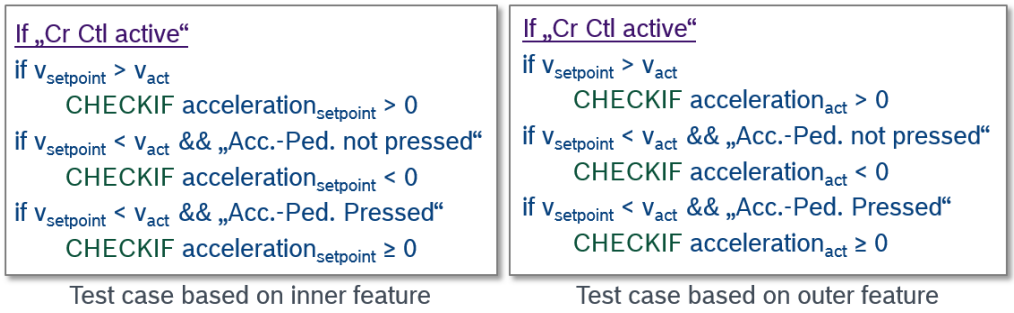

Fig. 5: Feature Test Cases based on Inner and Outer Feature Borders

3. Advanced Feature Testing

"Classic" testing is usually a sequential progression of test cases. Here an expected output is specified for a set of stimulating inputs. A test passes, if the actual output correlates to the expected output. Advanced Feature Testing on the contrary, focusses on parallel execution of test cases (see Figure 6). This allows evaluating multiple features at the same time and under exactly the same conditions. The Advanced Feature Testing test cases are only executed, if the specified conditions, like the activation of the corresponding feature, are met. Depending on the feature or rather the test case, it either consists of system state-independent checks or system state-dependent checks. The evaluation logic for system state-independent checks stays the same under all conditions. The evaluation logic for system state-dependent checks might vary depending on the actual system state. An example for a system state-independent check is as follows: "Exhaust system temperature must monotonical-ly increase".

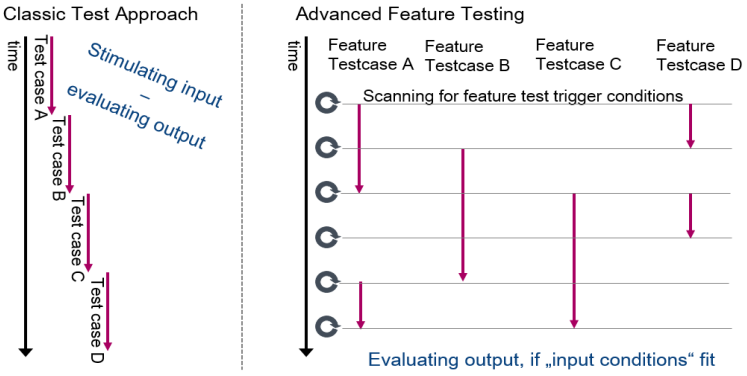

Fig. 6: Advanced Feature Testing vs. Classic Sequential Testing

Such test case could be activated, if feature "catalyst heating" becomes active. Examples for system state-dependent checks for feature "Cruise Control" are shown in Figure 5.

Figure 7 illustrates an example for the parallel execution of the feature test cases. Here for each feature, an actual value is compared to a lower and upper limit. As long as the specific feature value lies within this band, the test is passed. In the first third, all three features are active and the value to be checked lies within

the limit. In the second third, feature C becomes inactive, which apparently lead to its value changing abruptly. In the last third, all features are active. The variable of feature C crosses the band, which leads to a fault. One main advantage of this type of parallel execution of test cases is the reusability of test cases of the automatic evaluation of any feature combination.

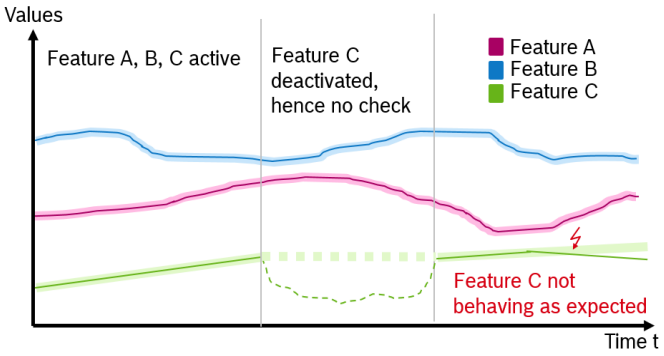

Fig. 7: Example for Advanced Feature Testing

4. Conclusion

Due to increasing number of features integrated in an automotive powertrain, testing those features and their interactions in an efficient way becomes a huge challenge. In this paper, we introduced two approaches for feature-testing of Engine ECU software: Feature-test-levels and Advanced Feature Testing. The introduction of feature-test-levels including the distinction of inner and outer feature borders is especially useful for non-feature-oriented SW structures. Advanced Feature Testing allows concurrent testing of features. This not only helps detecting feature dependencies, but also improves testing efficiency. Future work will focus the automation of Advanced Feature Testing.

5. Acknowledgements

Part of this research has been carried out in the FeatureOpt project (No. 849928), funded by the Austrian BMVIT (represented by the Austrian FFG).

6. References

[1] C. Kugler, S. Kowalewski, J. Richenhagen, R. Maquet, A. Schloßer. “Metrics-based strategies for quality assurance of automotive embedded software”. In: Bargende M., Reuss HC., Wiedemann J. (eds) 17. Internationales Stuttgarter Symposium. Proceedings. Springer Vieweg, Wiesbaden, 2017.

[2] P. C. Raut, S. L. Tade and R. S. Jadhav, "Test automation for Engine Warmup feature testing," 2016 International Conference on Computing Communication Control and automation (ICCUBEA), Pune, India, 2016, pp. 1-4.

[3] D. Ertl, S. Dominka and H. Kaindl, "Using a Mediator to Handle Undesired Feature Interaction of Automated Driving," 2013 IEEE International Conference on Systems, Man, and Cybernetics, Manchester, 2013.

[4] D. Wagner, H. Kaindl, S. Dominka, M. Dübner. “Optimization of feature interactions for automotive combustion engines”. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC '16). ACM, New York, NY, USA, 2016, pp. 1401-1406.

[5] P. Machado and A. Sampaio, “Automatic Test-Case Generation,” in Testing Techniques in Software Engineering: Second Pernambuco Summer School on Software Engineering, PSSE 2007, Recife, Brazil, December 3-7, 2007, Revised Lectures, P. Borba, A. Cavalcanti, A. Sampaio, and J. Woodcook, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 59–103.

[6] S. Apel. “Feature-Oriented Software Product Lines”. 1st ed. [S.l.]: Springer-Verlag Berlin An, 2016. Print.

[7] P. Machado, A. Vincenzi, and J. C. Maldonado, “Software Testing: An Overview,” in Testing Techniques in Software Engineering: Second Pernambuco Summer School on Software Engineering, PSSE 2007, Recife, Brazil, December 3-7, 2007, Revised Lectures, P. Borba, A. Cavalcanti, A. Sampaio, and J. Woodcook, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–17.