A Research Roadmap for Test Design in Automated Integration Testing of Vehicular Systems

Daniel Flemström
SICS Swedish ICT AB
Västerås, Sweden
danielf@sics.se

Thomas Gustafsson
Scania CV AB
Södertälje, Sweden
thomas.gustafsson@scania.com

Avenir Kobetski
SICS Swedish ICT AB
Kista, Sweden
avenir@sics.se

Daniel Sundmark
Mälardalen University
Västerås, Sweden
daniel.sundmark@mdh.se

***Abstract*—An increasing share of the innovations emerging in the vehicular industry are implemented in software. Consequently, vehicular electrical systems are becoming more and more complex with an increasing number of functions, computational nodes and complex sensors, e.g., cameras and radars. The introduction of autonomous functional components, such as advanced driver assistance systems, highlight the foreseeable complexity of different parts of the system interacting with each other and with the human driver. It is of utmost importance that the testing effort can scale with this increasing complexity. In this paper, we review the challenges that we are facing in integration testing of complex embedded vehicular systems. Further, based on these challenges we outline a set of research directions for semi-automated or automated test design and execution in integration testing of vehicular systems. While the discussion is exemplified with our hands-on experience of the automotive industry, much of the concepts can be generalised to a broader setting of complex embedded systems.**

Index Terms*—*Software Testing*; *Automotive Systems*; *Embedded Systems*; *Integration Testing

I. Introduction

Electrical systems in modern vehicles grow increasingly complex and software intensive. With additional concerns, like increasing requirements on autonomy and safety, integration- and system-level testing becomes more and more challenging. At full-vehicle integration level, in addition to the testing performed in actual vehicles, different types of electrical system lab testing is undertaken. Typically, in lab testing, use-case based functional (and to some extent non-functional) test cases are executed by means of hardware-in-the-loop (HIL) or software-in-the-loop (SIL) based integration testing platforms.

This current practice of test design and execution comes with a number of drawbacks. Test execution is costly and time-consuming, particularly if done manually. However, when test cases are scripted and automatically executed, testing tends to be repetitive and static. Moreover, the fact that test cases typically focus on one functional part at a time (without considering the potential interactions or interference between functions) may not account for realistic operating conditions.

In this paper, we address the question of whether it is possible to use automated test design, execution and analysis to test complex systems in general, and automotive and vehicular systems in particular, more effectively without exhausting the test resources. First, based on our experience with integration-level testing at a number of different vehicular Original Equipment Manufacturers (OEMs), we list a number of challenges that need to be addressed. Next, partially based on recent results in software and system testing, we outline a research roadmap for integration- and system-level testing of vehicular systems. In particular, we identify and describe five research directions for test design, automated test sequence generation and verdict analysis, that directly address the listed challenges.

II. Background

Over the last decades, embedded systems have been subjected to a rapid increase in complex functionality, and there are no indications suggesting that this trend will change in a foreseeable future. This is especially true for automotive systems, where emission regulations and *advanced driver assistance systems (ADAS)* energize the development. The functionality of ADAS depend on an increasing amount of sensor data. This leads to an increasing number of situations where human operators will no longer be in control. Based on input from various sensory sources, different functional components will interact and sometimes compete with each other and the operator. This adds flavour to the already non-trivial challenge of testing a product in its entirety.

The remainder of this section discusses state of the practice based on the collective experience from complex system integration testing in general, and integration-level testing of a number of vehicular OEMs (Volvo Construction Equipment (VCE), Bombardier Transportation (BT) and Scania) in particular. Although there are differences in particular details, the principles and challenges remain common.

A. State of the Practice

Today, testing is the primary means of assessing the quality of embedded systems. Testing is typically done at several stages throughout system development, ranging from unit-level testing of functions in isolation, to integration testing of fully interconnected systems. Ordered by the rising level of integration, testing is normally conducted in model-in-the-loop (MIL), SIL, HIL test environments, and full scale product tests. Typically, this is done through use-case based test sequences, partly derived from system requirements and partly reflecting

 ISBN: 978-1-61208-497-8

the test engineers' domain knowledge. This type of testing is commonly referred to as scenario-based testing [1].

In scenario-based testing, test cases are designed using a divide-and-conquer-based approach following the breakdown of system requirements into smaller functional entities (which is in accordance with recent textbook guidelines for functional test design [2], [3]). Test cases are typically constructed by means of sequences of (a) input or stimuli to the *system under test (SUT)*, (b) delays allowing the SUT to reach some desired state, and (c) assertions that compare expected behavior with the actual SUT behavior. Based on the outcome of its assertions, a test case, when exercised, can render three different verdicts: a *passed* test case is a test case with no violated assertions, a *failed* test case is a test case where the expected behavior does not match the behavior of the system under test (ideally indicating a fault in the target system), and an *aborted* test case is a test case where some test case action cannot be performed (ideally indicating a fault in the test case or the test environment). Once having been designed, implemented, and incorporated in the test suite, test cases are typically repetitively executed without much variation.

While manual testing is invaluable for quality assurance, especially when new functionality is introduced, it is expensive and not suitable for regression testing (i.e., following up on how software progresses over time whenever a new part of the system has been added to the SUT or an existing system has changed due to a bug fix or requirement update). For the latter purpose, test cases are—to an increasing extent—scripted to allow for automated batched execution. Analyzing which of the existing test cases should be executed in each regression test session is often a manual and time consuming process.

III. Current Practice: Limitations and Challenges

Although scripted scenario-based testing does ensure that requirements are exercised and covered during testing, it comes with a number of limitations. Below, we identify a set of challenges not addressed satisfactorily by current practice. It should be noted here that some manual testing practices (e.g., exploratory testing [4]) do address some of these challenges. However, the focus of this paper is on systematic automated techniques, primarily due to the fact that such techniques are more likely to scale with increasing complexity.

A. Ch1 - Lack of functional interference

The procedure of decomposing requirements into smaller functional units, and using these units in isolation as the basis for test case design is fundamental in managing an otherwise overwhelming complexity. However, this procedure for requirement decomposition, and subsequent test design also prevents the assessment of undesirable interference between functional entities. Such assessment is particularly important as the functions grow increasingly complex and dependent on several interacting subsystems. Small disturbances in one system component may propagate into fault conditions in another, not to mention the case when different functional entities are in conflict or competing over the same resource.

B. Ch2 - Inefficient resource usage

In order to make sure that each test has the right preconditions for making a correct verdict, test scripts are generally executed one by one, while the SUT's state is reset between the test scripts. In addition to the time overhead that is needed to set up and shut down each test case, delays are often encoded explicitly in test scripts to represent correct timing behavior (e.g., response times) of the SUT. Sequential test execution means that delay times can never run in parallel. Not only does this poorly represent real operating conditions, where several independent functions (e.g., cruise control, radio, and turn indicator lights) can be active at the same time, but it also leads to inefficient use of testing resources.

C. Ch3 - Inflexible and tedious test case encoding

Integration level test cases are often coded manually into a scripting language. One reason for this is the wish to keep a tight control over the order and timing of stimuli to the SUT to avoid incorrect test verdicts, i.e., verdicts (typically pass or abort) caused by incorrect test case implementation rather than the actual behavior of the SUT. In practice, this means that a test engineer needs to consider in detail not only what is the expected behavior for a certain function, but also how to put the SUT in a state where such behavior is supposed to be manifested. This mixes up two different views of the system, namely the design (and subsequent implementation) of test stimuli sequences and test oracles, making the task of a test engineer even more challenging. As a consequence, the development of integration test cases is a rather complex and time-consuming process, which severely limits the number of test cases that can be implemented and maintained.

D. Ch4 - Unnecessarily limited coverage

Any given functional requirement could in theory be tested by a large (if not infinite) set of concrete test cases. However, the current practice of hard-coding of scenarios into scripted test cases limits the potentials of variability in testing to only the hard-coded cases. Once designed and scripted, a scenario-based test case is typically kept and repetitively executed without much or any variation. While this has the advantage of detecting differences between software versions (e.g., for regression testing), it limits the testing to a very small portion of the vast set of imaginable real-life situations that the system could be subjected to, while the rest is left entirely unexplored. For example, the functionality of a hazard light could potentially benefit from being tested not only when cruising at low speed, but also in situations like: i) when the vehicle is idling, ii) while driving on a highway in winter conditions, iii) while applying brakes, etc. In practice though, most situations fall outside of the chosen set of test scenarios being considered important enough to encode and maintain.

E. Ch5 - Inadequate requirements

Previous work suggests that test cases are often based on positively stated requirements defining how the system should

 ISBN: 978-1-61208-497-8

behave during normal operation [5]. While this provides valuable confirmation with respect to the system's fitness for use in the normal case, there are results indicating that focusing on normal requirement-based cases might not be the best strategy when trying to maximize fault-detection [6], [7]. In fact, there exist many examples of accidents caused not by a software bug in the classical sense, but rather by incorrect or missing requirements, caused by a failure to consider important operational states or environmental conditions [8].

Also, a common problem in testing in general is that although some non-functional robustness testing is done, a disproportionally large part of testing is concerned mainly with functional requirements [9]. Related, there is a gap between requirements engineering (RE) and testing activities. In some cases, requirement documents are followed, but more often such documents must be complemented with the domain knowledge and understanding of what needs to be tested that the test engineer possesses. This is quite common in the industrial practice, also noted in, e.g., [9]. Another issue, which also depends on the above mentioned gap between RE and testing, is the practice of developing test cases for the same functionality independently, on different integration levels. As the development rate of new functionality is steadily increasing, such approach to system level testing seems insufficient.

F. Ch6 - The "smart product" challenge

In general, embedded products become increasingly intelligent, while testing methods are lagging behind. In the vehicular world, this is best highlighted by ADAS functionality, which introduces additional challenges to the software development process and to testing in particular. Advanced algorithms are to an increasing extent replacing human decision making. While human drivers might and do make mistakes, there is close to zero-tolerance when it comes to autonomous functions doing so. Thus, ADAS-type functions must be tested even more extensively. Also, their interactions with other functionality in the vehicle, as well as with the driver, must be taken into consideration. Most ADAS functionality is triggered by the surrounding environment and the different situations that a vehicle may be subjected to. This is a challenge in a laboratory setting (e.g., HIL), where the environment must be represented in some way.

IV. Research Directions

Considering the above stated challenges, we believe that much can be gained from moving beyond the prevailing script-based integration test design and execution practice, which is primarily focused on one single function or even a part of a function (use case) in isolation. In general, we envision a situation where automatically executable test sequences are generated or derived in a semi-automated or fully automated fashion. These test sequences are derived based on high-level test design rationales that address the above challenges. The sequences can be automatically executed and their verdict can be automatically analysed and reported. Below, we list a number of research directions that serve to push the integration testing of vehicular systems towards this vision.

First, considering the inflexible and tedious test case encoding challenge (Ch3), we suggest to draw a clear line between generation of test stimuli sequences (i.e., timed and ordered sequences of input data to be fed to the SUT), and encoding of expected test responses (i.e., how the SUT should respond given a certain sequence of events or stimuli). This research direction is elaborated in the subsection on **Separation of Concerns**. Second, concerning *test sequence generation (TSG)*, we identify three distinct rationales that directly address challenges Ch1, Ch2, Ch4, and Ch6. These rationales are **Functional Interference**, focusing on the extent to which the interaction between features is covered during testing, **Environmental Coverage**, focusing on the extent to which aspects of the intended environment of the vehicle are covered during testing, and **Diversity**, focusing on the extent to which test cases and test suites are different from one another. Each of the above listed research directions are discussed in detail below. Third, modeling of expected responses has as its goal to robustly assess whether the response of the SUT to a variety of (automatically generated) test sequences is adequate (i.e., should yield a *pass* verdict) or not (i.e., should yield a *fail* verdict). In software testing, this problem is known as **the Oracle Problem** [10], and we address it in a separate subsection below. Referring to Ch5, note that any requirements that should be testable need to be reflected in test oracle models in an adequate way.

A. Separation of Concerns

As mentioned in challenge Ch3, the traditional way of encoding stimuli and verdicts into one executable unit makes the resulting test cases both less flexible and more difficult to analyze. Consequently, we believe that it is important to clearly separate: a) the question of how to combine test stimuli into effective and feasible test sequences, and b) the question of which test assertions to make in order to produce a test verdict, and when to evaluate these assertions.

With such separation in place, test stimuli modeling can be seen as a special case of *model-based testing (MBT)*, where the modeling scope, according to Utting et al.'s taxonomy [11], is limited to input-only. In other words, when reasoning about TSG, there is no explicit need to consider the test output, i.e., the actual verdict analysis. Instead, one can focus on questions such as what kind of input should be considered for guiding SUT through its possible states, and how this input should be modeled and selected into the actual test sequences to achieve effective and efficient testing.

Considering the other side of the problem, i.e., how to reach a test verdict given an automatically generated test sequence, with this approach test engineers can focus more explicitly on exactly which preconditions are needed to trigger a test oracle function, and which results it should produce under different conditions. While appropriate models need to include both input and output aspects, the input part does not actually drive the state progression. Rather, it describes the state(s) in

 ISBN: 978-1-61208-497-8

which the SUT needs to be in order to perform meaningful test evaluations. Ideally, such models should clearly reflect the requirements on functionality.

While separation of concerns contains relatively few research questions in itself, it is central not only for reducing test case generation complexity, but also for research on each of the separated parts. An important research direction here is to find appropriate interfaces between the separated parts, which in turn influences the choice of modeling formalisms in the two distinct cases. Some preliminary results have already been pointed to in this direction [12], but much work remains.

B. Functional interference

In real operating conditions, functional interference is the norm rather than an exception [13]. Different parts of a complex product's functionality are typically being engaged simultaneously and independently from each other. Taking an automotive application as an example, a lane shift may involve both turn indicator and acceleration functions. At the same time, the climate control system may be trying to keep temperature at some set point. Including ADAS into consideration, e.g. blind spot assist or adaptive cruise control functionality may also be active during the lane shift.

One goal of integration testing is to test how different functions work together. However, use-case based testing typically focuses on one function at a time, and the concurrent activation of other functions is only a side-effect of reaching a testable system state. Clearly, if interactions between functions are not *systematically* tested, there is a risk of missing important errors. In fact, function or subsystem interactions are responsible for a growing portion of accidents in complex systems [8]. Also, it is important to carefully consider possible interactions between autonomous functionality and human actions. In fact, in fields where automation was adopted early, such as aviation, such interactions are known to contain a certain risk [14].

Consequently, it is desirable to allow for testing of several functions run in parallel. In the ideal case that any combination of functions that in some way affect one another or a common functional or non-functional resource was tested together, Ch1 could be removed from the above list of challenges. Conversely, the usage of testing resources (Ch2) would be much more efficient if all fully independent function combinations could be batched together into a single run. However, there are a number of questions to solve before reaching that situation. For example, what is an adequate runtime environment for parallel testing of potentially dependent functions? Possibly, ideas could be drawn from the field of parallel computing. However, they should be adapted to the specifics of integration testing of embedded systems, e.g. complex functional dependencies, hard real-time constraints, wide range of different users and operational contexts, safety criticality, etc., see also [13].

Since testing resources are normally limited, the right mix between interacting and independent functionality will likely be an important design trade-off that should be considered by TSG algorithms. Also, infeasible combinations of functions, i.e., those leading to test abortion, must somehow be avoided.

Different types of functionality need to be modeled in a suitable way, such as actuator-triggered (driver controls), sensor-triggered (autonomous responses), and failure-triggered.

C. Environmental coverage

An important factor that affects the operation of embedded systems, often neglected in scenario-based testing, is the impact of the surrounding environment on a system's performance and operation. This aspect is important not only by extending the notion of test coverage with a new dimension (thus addressing Ch4), but it increases in significance with the growing "smartness" of embedded products (Ch6). The more autonomous functionality a system contains, the more important it is to anticipate and test possible situations that the system can be subjected to. This need has recently been formulated by Alexander et al. [15], as a situation coverage metric for autonomous systems.

The idea is to describe real-life situations by partitioning them into a number of constituent components, or environmental aspects, each of which consists of a number of discrete (and typically mutually exclusive) elements, or possible values. Returning to the automotive world for an example, such components could be the topography of the road (uphill, downhill, or negligible inclination), road surface conditions (snow, ice, rain, dry), surrounding traffic (pedestrians, queues, highway, platooning, etc.), intersection types (3-road, right-turn, left-turn, straight driving, etc.), driver condition (alert, tired, using phone, etc.), and so on.

Once the environmental components have been identified, they can be combined into more or less complex situations and tested together with several active functional elements. For example, a vehicle can be driving uphill on a snowy highway, conducted by a sleepy driver that makes a lane shift and presses down the acceleration pedal, while blind spot assist and adaptive cruise control functionalities are activated.

Evidently, situation coverage poses similar research questions as the ones discussed in the previous subsection, perhaps the most obvious being the choice of appropriate environmental models, with respect to, e.g., abstraction level, modeling formalism, TSG tools, etc. Further, since the number of possible combinations of situational components and functions seems to suffer from combinatorial explosion, it is not realistic to believe that every possible aspect will be tested in each test run. Thus, combinatorial strategies on selecting relevant situations with respect to the testing objectives are needed [16].

D. Diversity

A recent research direction in software testing investigates the effects on fault detection and coverage of the extent to which test cases are different from each other. In particular, *diversity* metrics based on information theory have been used for this purpose [17]. Diversity is typically defined as a metric between 0 and 1 indicating the distance between two test cases, or sets of test cases. Several studies indicate that increased test case diversity has a positive effect on the ability to discover faults. For instance, Mondal et al. [18] used diversity as a

 ISBN: 978-1-61208-497-8

complement to traditional test adequacy criteria and found that combining diversity with other criteria yields better fault detection rates. Feldt et al. [17] define a diversity metric for sets of test cases, which allows for search-based selection methods to work on entire sets of test cases. There are however several different definitions of diversity, and thus, ways of measuring it. Examples of such metrics are based on the normalised compression distance (NCD) between the textual representation of a variability model of the test cases [17], [19], the euclidian distance between input/output vectors [20] or features thereof [21], and weighted combinations of metrics for different properties of the measured items [22].

Extending the concepts of e.g test input/output diversity, we may address the first challenge, Ch1, by using diversity as a criterion to ensure that a generated set of test cases involves as different functions as possible. Further, using diversity measures to guide the selection of test cases between regression testing sessions would also address Ch4 to avoid that the same, or too similar, tests are being executed from time to time, thus yielding a better coverage over time.

Given the promising results in the mentioned studies (although largely focused on unit testing), diversity stands out as an attractive optimization criteria for a test stimuli generator as discussed in Section IV-A. There are however numerous challenges with this approach, including the level of detail of the available information and to what extent such information can be efficiently retrieved. Further challenges include how to apply diversity in an event-based testing environment (e.g., considering timing and parallelism), and how to best combine diversity with other metrics (e.g. requirements, environmental coverage, or functional interference). Research applicable definitions of diversity that are effective on integration level with respect to the different challenges listed in Section III is therefore needed. Finally, the computational effort when calculating the diversity measures needs to be addressed in order to be suitable for a test sequence generator.

E. Test oracle modeling

A *test oracle* is a mechanism that, given a certain input and the SUT's response to that input, can state whether this actual response is in accordance with the expected response (i.e., if the test passes or not). In software testing, the construction of such a mechanism is known as the oracle problem. The oracle problem is relatively unexplored and inherently difficult to address [10]. In practice, since complete oracles would require an exhaustive and correct representation of the expected behaviour of the SUT, only partial oracles are possible to construct, typically encoded as assertions in test cases.

As mentioned above, we believe that test oracles should be modeled as passive analysis mechanisms, expressing how a SUT should behave given a certain sequence of stimuli. Condition models [23] and guarded assertions [24] are two examples of initial attempts to address this problem. However, further development is needed to reach practical applicability.

Ideally, oracle models should be formal enough to allow for automatic translation into test code. Further, they should have clear and human-readable links to the functional requirements they represent. This would promote requirement traceability and ability to reason about logical relations between tests and requirements, reducing the gap between these disciplines (Ch5). Also, automation of the logic behind oracles should then be possible, supporting oracle analysis either online, e.g., by testing in a HIL environment, or offline, e.g., by a verification algorithm. Balancing between informal and formal requirement models is thus an intriguing research topic.

Also, human-readable models should be developed early on, allowing their reuse through different stages of the testing process. Modeling patterns on different abstraction levels, and unambiguous conversion between the different models and the test code will likely be needed for any such approach to be applicable. Related, structured English grammars, together with patterns to facilitate writing requirements, suitable for automated checking of system and requirements conformance, have been proposed in several papers [25], [26], [27].

Finally, addressing the remaining parts of Ch5, oracle models reflecting non-positive or non-functional requirements will be needed. The challenge is further complicated by the immaturity of the RE field in this respect. There is a clear need for research both on basic RE in this direction, and next on enriching the results of such basic research into the more applied question of oracle modeling.

F. Summary

Above, five research directions are outlined for test design in integration testing of vehicular systems. Considering all these research directions combined, one could envision the following integration test design, execution and analysis process: **First**, requirements are (manually) encoded as abstract and passive test oracles using human readable and intuitive patterns. **Second**, based on these oracles, but also considering other test design rationales like functional interference, diversity and/or environmental coverage, test stimuli sequences are automatically generated. **Third**, the test sequences are executed on the system under test, and the oracles automatically analyze the system response in order to produce an aggregated test verdict.

Naturally, the outlined process is only one possible way of moving forward and should be revised as testing research and practice progress. Further, it is important to relate the industrial experience, to recent academic advances, drawing inspiration from the broader test research field, e.g. [28], [29].

V. CONCLUSIONS

In this paper, a number of challenges facing the discipline of integration testing is outlined, based on the authors' experiences from industrial vehicular systems. The increasing autonomy and complexity of modern vehicles, which leads to a high level of functionality interaction, and in consequence complex emergent behavior, need to be accounted for in integration testing. This poses an additional burden on the already restrained testing resources. In addition, it becomes more difficult to reason about system behavior, which makes coverage an even more important aspect to consider.

 ISBN: 978-1-61208-497-8

The challenges are considered in a discussion on interesting research directions. Firstly, as in other areas of software engineering, concerns should be separated where possible. It is our claim that generation of test stimuli sequences can and should be separated from the actual oracle function. This will allow a focus on the different parts separately, generating appropriate models for each. On the oracle side, it is important to capture requirements at the right level of abstraction, ideally transforming them into more formal models that can be verified by a test run. When it comes to test stimuli, there is a need for appropriate models that capture actual situations to which a vehicle can be subjected, with several interacting functions being active simultaneously.

An important topic for research is how to increase the variability of what is tested. One answer to this question is to make the test input space as diverse as possible. However, vehicles are complex creatures operating in complex environments, and so the modelling of possible inputs becomes challenging, not to mention the question of what is meant by diversity and how to select appropriate test stimuli in order to produce a diverse test suite. Taking functional interaction and coverage of environmental conditions into consideration may provide some answers to the above questions.

ACKNOWLEDGMENT

This work was supported by The Swedish Innovation Agency (Vinnova) through grant 2015-04816, and the Swedish Knowledge Foundation through grant 20130258.

REFERENCES

[1] A. Bertolino, E. Marchetti, and H. Muccini, "Introducing a reasonably complete and coherent approach for model-based testing," *Electr. Notes Theor. Comput. Sci.*, vol. 116, pp. 85–97, 2005.

[2] M. Young and M. Pezze, *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley & Sons, 2005.

[3] P. Ammann and J. Offutt, *Introduction to Software Testing*, 1st ed. New York, NY, USA: Cambridge University Press, 2008.

[4] J. Itkonen, M. V. Mäntylä, and C. Lassenius, "The role of the tester's knowledge in exploratory software testing," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 707–724, May 2013.

[5] L. M. Leventhal, B. Teasley, D. S. Rohlman, and K. Instone, "Positive test bias in software testing among professionals: A review," in *Selected papers from the Third International Conference on Human-Computer Interaction*, ser. EWHCI '93. London, UK, UK: Springer-Verlag, 1993, pp. 210–218. [Online]. Available: http://dl.acm.org/citation.cfm?id=646181.682601

[6] L. M. Leventhal, B. E. Teasley, and D. S. Rohlman, "Analyses of factors related to positive test bias in software testing," *Int. J. Hum.-Comput. Stud.*, vol. 41, no. 5, pp. 717–749, Nov. 1994. [Online]. Available: http://dx.doi.org/10.1006/ijhc.1994.1079

[7] A. Causevic, R. Shukla, S. Punnekkat, and D. Sundmark, "Effects of negative testing on tdd: An industrial experiment," in *International Conference on Agile Software Development, XP2013*, H.Baumeister and B. Weber, Eds. Springer, June 2013, Date accessed: 2016-06-09. [Online]. Available: http://www.es.mdh.se/publications/2771-

[8] N. G. Leveson, "System safety in computer-controlled automotive systems," *SAE transactions*, vol. 109, no. 7, pp. 287–294, 2000.

[9] Z. A. Barmi, A. H. Ebrahimi, and R. Feldt, "Alignment of requirements specification and testing: A systematic mapping study," in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*. IEEE, 2011, pp. 476–485.

[10] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, May 2015.

[11] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[12] T. Gustafsson, M. Skoglund, A. Kobetski, and D. Sundmark, "Automotive system testing by independent guarded assertions," in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, April 2015, pp. 1–7.

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

[14] T. B. Sheridan and R. Parasuraman, "Human-automation interaction," *Reviews of human factors and ergonomics*, vol. 1, no. 1, pp. 89–129, 2005.

[15] R. Alexander, H. Hawkins, and A. Rae, *Situation coverage – a coverage criterion for testing autonomous robots*. Department of Computer Science, University of York, 2 2015, vol. Report number YCS-2015-496.

[16] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 11:1–11:29, Feb. 2011. [Online]. Available: http://doi.acm.org/10.1145/1883612.1883618

[17] R. Feldt, S. Poulding, D. Clark, and S. Yoo, "Test set diameter: Quantifying the diversity of sets of test cases," *arXiv preprint arXiv:1506.03482*, 2015.

[18] D. Mondal, H. Hemmati, and S. Durocher, "Exploring test suite diversification and code coverage in multi-objective test case selection," in *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 1–10.

[19] R. Feldt, R. Torkar, T. Gorschek, and W. Afzal, "Searching for cognitively diverse tests: Towards universal test diversity metrics," in *Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on*. IEEE, 2008, pp. 178–186.

[20] P. Bueno, W. E. Wong, and M. Jino, "Improving random test sets using the diversity oriented test data generation," in *Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*. ACM, 2007, pp. 10–17.

[21] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Simcotest: a test suite generation tool for simulink/stateflow controllers," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, 2016, pp. 585–588, Date accessed: 2016-06-09. [Online]. Available: http://doi.acm.org/10.1145/2889160.2889162

[22] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "Object distance and its application to adaptive random testing of object-oriented programs," in *Proceedings of the 1st international workshop on Random testing*. ACM, 2006, pp. 55–63.

[23] A. Ray, I. Morschhaeuser, C. Ackermann, R. Cleaveland, C. Shelton, and C. Martin, "Validating automotive control software using instrumentation-based verification," in *Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on*. IEEE, 2009, pp. 15–25.

[24] G. Rodriguez-Navas, A. Kobetski, D. Sundmark, and T. Gustafsson, "Offline analysis of independent guarded assertions in automotive integration testing," in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*, Aug 2015, pp. 1066–1073.

[25] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (ears)," in *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*. IEEE, 2009, pp. 317–322.

[26] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, and A. Tang, "Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar," *Software Engineering, IEEE Transactions on*, vol. 41, no. 7, pp. 620–638, July 2015.

[27] P. Filipovikj, M. Nyberg, and G. Rodriguez-Navas, "Reassessing the pattern-based approach for formalizing requirements in the automotive domain," in *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*. IEEE, 2014, pp. 444–450.

[28] M. J. Harrold, "Testing: a roadmap," in *Proceedings of the conference on the future of software engineering*. ACM, 2000, pp. 61–72.

[29] A. Orso and G. Rothermel, "Software testing: a research travelogue (2000–2014)," in *Proceedings of the on Future of Software Engineering*. ACM, 2014, pp. 117–132.

 ISBN: 978-1-61208-497-8