CHALMERS

Evaluation of validity of verification methods

Master of Science Thesis
Project planning report

Oskar Ingemarsson
Sebastian Weddmark Olsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden, September 2013

Evaluation of validity of verification methods

OSKAR INGEMARSSON
SEBASTIAN WEDDMARK OLSSON

Examiner: JOSEF SVENNINGSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

{Cover:
an explanatory caption for the (possible) cover picture
with page reference to detailed information in this essay.}

Department of Computer Science and Engineering
Göteborg, Sweden, September 2013

Contents

Chapter 1

Introduction

In the first sections of this chapter we outline the background, purpose and objectives and the last section describes the scope of our research. Methods is described in Chapter 2 and a time-plan exists in Chapter 3.

1.1 Background

The last decade of the 20th century resulted in a dramatic growth of technology [1]. The rate is still increasing with up to 90% of all innovations being realized through electronics in the beginning of the 21st century [2], and over 80 % of all innovations in the automotive industry is in the electronics and software area [3]. The software-based systems in motor vehicles have become more complex, and are moving toward handling more critical functions [3]. Vehicles have already begun to communicate with each other [4], and it is soon expected that roadside traffic management systems will also interact with the systems in vehicles [5].

While the systems become more complex the software must be fault-tolerant and safe. Testing, validation and verification is a need and should follow all product development phases, from start to finish. The problem is that testing is both time consuming and labour intensive, accounting for up to 50 % of the development cost [6]. Unit tests adds additional complexity to the code. It is very important to test and verify all steps of the development.

The complexity issue is that in systems such as microprocessors the number of possible failure modes is so large it is considered infinite [7]. This makes it impossible exhaustively test the system, and therefore, also make the detection of failures unreliable.

Quviq QuickCheck has the ability to automate this process, and allow the developer to write properties instead of tests. These properties make it possible for QuickCheck to create arbitrary input vectors that can be feed to the code. Figure 1.1 shows the different steps of the software development process and the test and verification phases it has.

The first step is the system design. At this point the system specification is written, and then the software specifications. When all specifications exists, the next phase is to

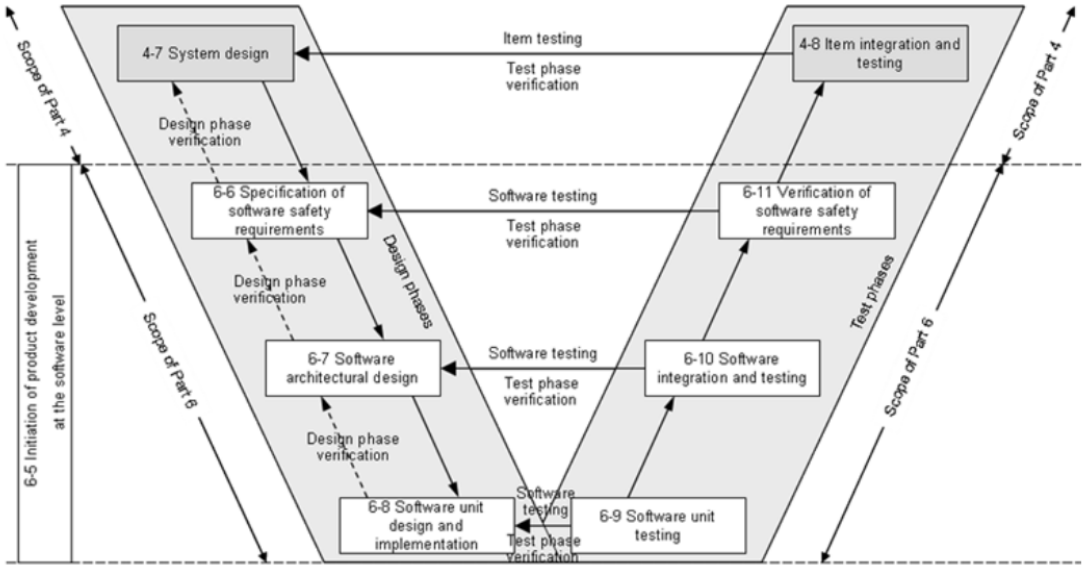


Figure 1.1: Phase model for software development

design the software architecture. The last part of the design is software unit design and implementation. All phases must be a verification of the former phase.

When the implementation is done, it is time for the test phases. These begins with software unit testing which tests the software unit design phase. If the tests verifies that the software unit design and implementation is correct, the test phase moves on to the architectural design and then to verification of software safety requirements. The last test phase verifies that the system is designed according to the specification.

1.2 Purpose

The purpose is to automate the testing process in an effective and good way. To make it possible to raise the Automotive Safety Integration Level (ASIL), in the software unit design and implementation phase and in the software architectural design phase, a check must be done to see if it exist a tool that can be used in order to perform a semi formal verification of a module, and then make this process generalized for modules in coexistence.

The purpose is also to be able to decrease the number of needed unit tests in the software unit design and implementation phase. Even further is the goal to introduce semi formal verification of the software architectural design phase. The functional safety concept will be most important in this phase.

1.3 Objective

The objective is to propose and motivate what should be done to be able to achieve a semi formal verification. This should include a confidence interval for how certain the verification is.

The objective is also to prove that it is possible to do semi formal verification for an AUTOSAR module and its specification. This should be generalized so its possible to test other specifications and modules at a later moment. It should also not matter which configuration that is active, because the specification should hold for all configurations.

1.4 Scope

We will use AUTOSAR 4.0 revision 3 for our thesis work. Since this version of AUTOSAR consists of around 80 specifications and other auxiliary materials [8], we will limit our scope to one or two specifications. The main module of this thesis is the CryptoServiceManager. This module provides cryptographic functionalities for synchronous and asynchronous services. This module is chosen because it only got a few dependencies, is used to trace development and production errors but also to incorporate cryptographic libraries.

It is hard to test that a call to another module gives the right results. This is the reason why we have chosen a module with a small number of dependencies. The CryptoServiceManager should also have functionality which gives the same results no matter which state it is in, such as hash functions [9].

Also the aim of the work is to verify software components. In other words no work considered hardware or a combination of hardware and software will be prioritized. All implemented code for the verification will run on a standard PC-machine.

Chapter 2

Method

2.1 Choose of tool for verification

Software unit testing can be achieved by almost any tool. Consequently this phase is not the most interesting when it comes to the choose of a tool verification. Of course one can take the simplicity to achieve good unit testing into account, but still it is not what makes a verification tool especially unique for the project goals.

Since the purpose is about benchmarking software the phase "verification of software safety requirements" will not influence the choose. To be able to test this phase, a greater amount of components of the whole system must be available. Such components may include hardware etcetera. Implementation wise should everything be able to run on a standard PC-machine.

The most interesting part is the software integration and testing. Is there a tool that one can use to easily combine test and requirements from different modules? Is it possible to test functional safety concept from this combination, for example by corrupting some software elements?

2.2 Specification

In AUTOSAR, specifications for each module is given in text form. Consequently one must first, before a module can be tested, implement the specification for that module in code.

2.3 Testing

Properties for a module have to take the current state in consideration, since most functions written in an imperative language are not immutable. This gives raise to the idea of a state based testing tool.

- Choose a specification which will be translated to QuickCheck properties in parts.

- With the use of statistics and confidence intervals, show that, with enough tests the state-space will be exhausted.
- Evaluate other semi formal techniques and show that the results from them shows that QuickCheck is reliable for verification.
- Generalize the technique.

2.4 Implementation

A challenging step is the analysis of the results. If the testing tool returns zero errors what does that say about the robustness of the input byte code? Passed 100 of 100 tests is just a statement and does not say anything more than that some tests passed. Can tests be implemented in a clever way so that it is possible to get some kind of confidence interval on the correctness of the code?

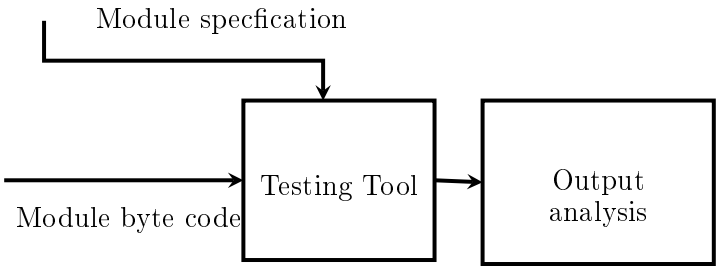


Figure 2.1: Abstract implementation module

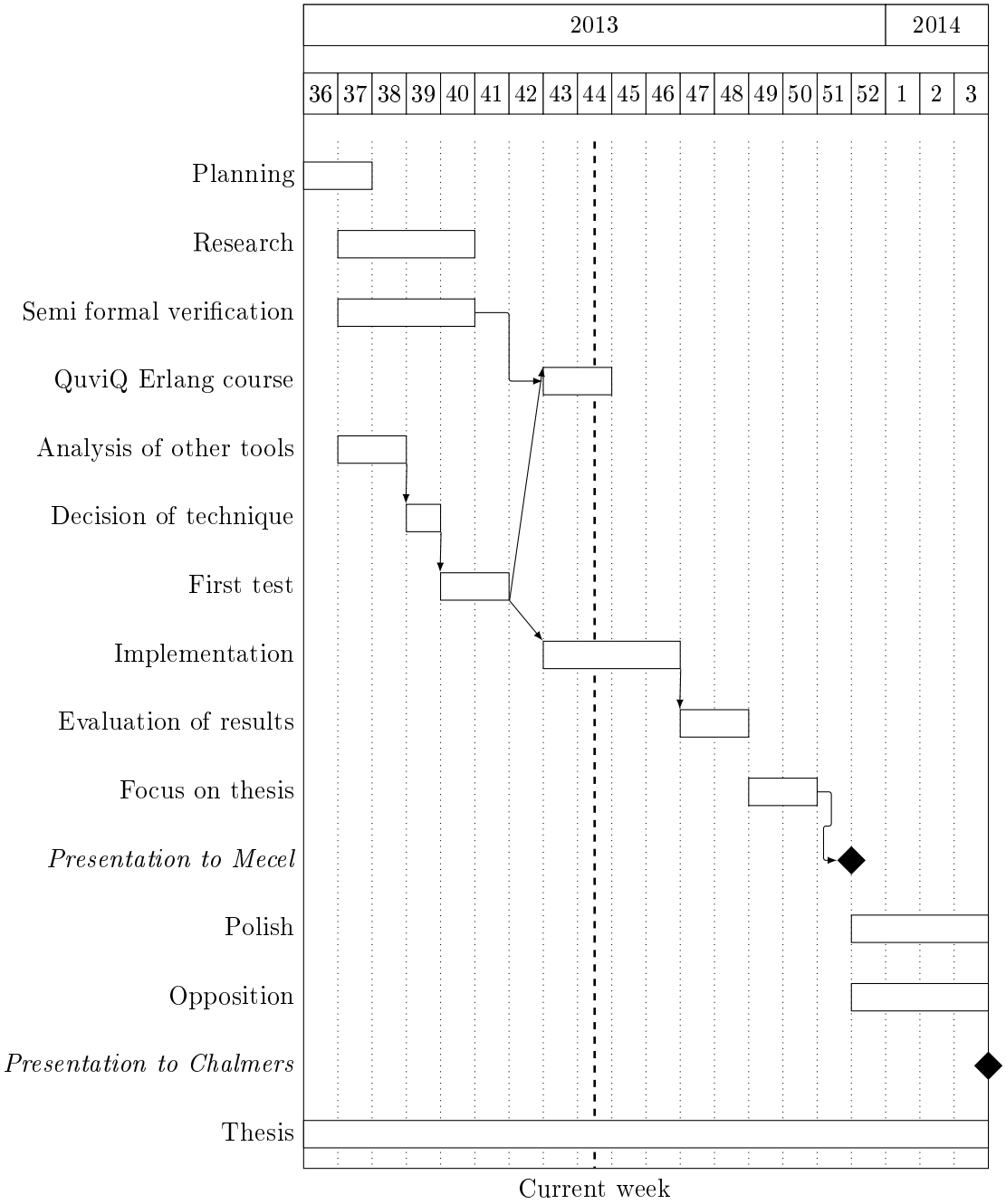
Chapter 3

Time-plan

3.1 Provisional plan

Exactly how things should be done isn't clear. Therefor we are proposing a more orderly plan to be able learn what can be done and where problems may lie a head. At first just try to implement one or two simple rules from an AUTOSAR specification, in Erlang, and try to run it on AUTOSAR module. It's important to notice that this may have nothing to do with our final solution and is mostly for evaluating what can be done in QuickCheck. Since our work is not just about implementing tests, but rather show that we can make semi formal verification on a module in AUTOSAR, the number of rules that we implement seems less relevant than what we can do with the rules that we implement.

The next step is to analyse the results and come to a conclusion for how to continue. Possible Erlang and QuickCheck is not suited for our goals. More information about this can be found on the Gantt scheme represented below.



Finding out the exact definition of semi formal verification and getting an idea of how this can be implemented is a fundamental step. Hence this is added explicitly, as shown in the Gantt Scheme, to the project plan. In the first test week we translate some specification demands to QuickCheck code, and test if it is possible to prove that this is semi formal verfied. The purpose of doing analysis of other tools is to see if there exists better tools for our goals, we should then decide which tools and techniques to use. A

presentation to Mecel is planned to the last week in December. In January we present our work to Chalmers.

Bibliography

[1] Gilberg A. AUTOSAR (Automotive Open System Architecture) A Key Technological Enabler for ECUs integration; 2011. AUTOSAR Spokesperson (PSA Peugeot Citroën). 2nd International Conference on Chassis Electrification 2011, May 13, Frankfurt, Germany.

[2] Trage S. Electronics: Driving Automotive Innovation; 2004. Frost & Sullivan.

[3] Voget S. Interoperability as prerequisite for functional safety in the automotive industry. Siemensstraße 12, 93055 Regensburg: Continental Automotive; 2012.

[4] SARTRE. Safe road trains for the environment;. `http://www.sartre-project.eu/en/Sidor/default.aspx`.

[5] Strandén L, Ström E, Uhlemann E. Wireless Communications Vehicle-to-Vehicle and Vehicle-to-Infrastructure. SP, Chalmers and Volvo Technology; 2008.

[6] Claessen K, Hughes J. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs; 2000.

[7] Storey N. Safety-Critical Computer Systems. ISBN 978-0-201-42787-5. Prentice-Hall; 1996.

[8] AUTOSAR. AUTOSAR; 2013. `http://autosar.org/index.php?p=3&up=2`.

[9] AUTOSAR. Specification of Crypto Service Manager;. Document Identification Number 402.