Mälardalen University Licentiate Thesis 257

SIMILARITY-BASED TEST EFFORT REDUCTION

Daniel Flemström

MÄLARDALENS HÖGSKOLA
ESKILSTUNA VÄSTERÅS

Mälardalen University Press Licentiate Theses
No. 257

SIMILARITY-BASED TEST EFFORT REDUCTION

Daniel Flemström

2017

School of Innovation, Design and Engineering

ISBN 978-91-7485-318-6
ISSN 1651-9256
Printed by E-Print, Stockholm, Sweden

To Eva

Sammanfattning

Inbyggda datorsystem finns överallt omkring oss idag; i allt från diskmaskiner till bilar och flygplan. Ofta finns det stränga krav på såväl korrekt funktion som svarstider. Eftersom programvaran i det inbyggda systemet i till exempel en bil är väldigt stor och komplex, utvecklar man programvaran i mindre delar som sedan successivt integreras till det färdiga datorsystemet. Denna del av utvecklingsprocessen kallas systemintegrationsfasen och är en av de största och mest kritiska faserna när man utvecklar inbyggda system. Integrationen kan utföras i ett antal nivåer med utförliga tester av mjukvaran på varje nivå. Detta innebär att det krävs ett avsevärt arbete för att testa mjukvaran. Om man kan minska detta arbete, givetvis utan att ge avkall på testningens kvalitet, förväntas det få en stor effekt på det totala utvecklingsarbetet.

Studier har visat att en icke försumbar del av testarbetet är slöseri på grund av att människor, omedvetet eller av nödvändighet, utför likartade (eller till och med överlappande) testaktiviteter. En konsekvens är att testfall riskerar att bli helt eller delvis lika. Vi identifierade sådana likheter i en fallstudie med 2500 manuella testfall, skrivna på vanligt språk, från fyra projekt inom fordonsindustrin. Information om likheter och överlapp kan användas för att, till exempel, minska arbetsåtgången vid underhåll eller när man översätter testfall till kod så att de kan utföras automatiskt av en dator (automatisering).

I en annan studie inom samma domän, undersökte vi flera metoder för att prioritera arbetsordningen vid automatisering av testfall där liknande testfall kunde återanvändas. I studien analyserade vi hur denna ordning påverkar arbetsmängden för fyra industriprojekt med totalt 3919 integrationstestfall skrivna på vanligt språk. Resultaten visar att den bästa ordningen kan minska testarbetet avsevärt. En förutsättning för detta är att redan översatta delar av testfall kan återanvändas för att slippa översätta liknande testfall igen.

En annan väg för att minska testarbetet är att återanvända testfall och information mellan integrationsnivåer. Vi har kartlagt metoder och motiv kring sådan återanvändning i en systematisk mappningsstudie. Våra fallstudier visar vidare att det både är genomförbart och lönsamt att hålla reda på likheter i testfallen. Slutligen konstaterar vi att arbetsinsatsen för manuell testning påverkas av automationsordningen när det är möjligt att återanvända redan översatta delar av liknande testfall.

Abstract

Embedded computer systems are all around us. We find them in everything, from dishwashers to cars and airplanes. They must always work correctly and moreover, often within certain time constraints. The software of such a system can be very large and complex, e.g. in the case of a car or a train. Hence, we develop the software for embedded systems in smaller, manageable, parts. These parts can be successively integrated, until they form the complete software for the embedded system, possibly at different levels. This phase of the development process is called the system integration phase and is one of the most critical phases in the development of embedded systems. In this phase, substantial effort is spent on testing activities.

Studies have found that a considerable amount of test effort is wasteful due to people, unknowingly or by necessity, performing similar (or even overlapping) test activities. Consequently, test cases may end up to be similar, partially or wholly. We identified such test similarities in a case study of 2500 test cases, written in natural language, from four different projects in the embedded vehicular domain. Such information can be used for reducing effort when maintaining or automating similar test cases.

In another case study in the same domain, we investigated several approaches for prioritizing test cases to automate with the objective to reduce manual test effort as quick as possible given that similar automated tests could be reused (similarity-based reuse). We analyzed how the automation order affects the test effort for four projects with a total of 3919 integration test cases, written in natural language. The results showed that similarity-based reuse of automated test case script code, and the best-performing automation order can reduce the expected manual test effort with 20 percentage points.

Another way of reducing test effort is to reuse test artifacts from one level of integration to another, instead of duplicating them. We studied such reuse methods, that we denote vertical reuse, in a systematic mapping study. While the results from of our systematic mapping study showed the viability of vertical test reuse methods, our industrial case studies showed that keeping track of similarities and test overlaps is both possible and feasible for test effort reduction. We further conclude that the test case automation order affects the manual test execution effort when there exist similar steps that cannot be removed, but are possible to reuse with respect to test script code.

Acknowledgments

A lot of people have supported me on the journey that resulted in this very thesis. I realize that I have been very fortunate to have the luxury of a main supervisor (Professor Daniel Sundmark) actively participating in all meetings and papers and a co-supervisor (Dr. Wasif Afzal) that has been curling me beyond expectations. Further, I am utterly thankful for the enthusiastic and energizing support from my newest supervisor: Dr. Eduard Enoiu. Without you, this path would have been much more dull and at least a decade longer. I also have been very fortunate to have such a supportive and knowledgeable main industrial contact as Ola Sellin.

Of course, I would not forget to mention the people at Bombardier Transportation Sweden AB in Västerås, RISE SICS, RISE SICS Västerås AB, Mälardalen University, Volvo Construction Equipment and ABB HVDC that have contributed with their kind support and fruitful discussions.

Above all, I realize is that I am also very fortunate to still have my family, considering me as a family member, despite many too long working hours and neglect.

Finally, I would like to thank the Swedish Governmental Agency of Innovation Systems (Vinnova) Mälardalen University, Volvo Construction Equipment, Bombardier Transportation Sweden AB and ABB HVDC whose support via the IMPRINT project, has made this thesis possible.

Västerås, April 2017

List of Publications

Appended Studies

This thesis is based on the following studies:

Paper A: Vertical Test Reuse for Embedded Systems: A Systematic Mapping Study. Daniel Flemström, Daniel Sundmark and Wasif Afzal. Published at The 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA'15).

Paper B: Exploring Test Overlap in System Integration: An Industrial Case Study. Daniel Flemström, Wasif Afzal and Daniel Sundmark. Published at The 42nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA'16).

Paper C: Similarity-Based Prioritization of Test Case Automation. Daniel Flemström, Pasqualina Potena, Daniel Sundmark, Wasif Afzal, Markus Bohlin. (Submitted to the Special Issue of Software Quality Journal on Automation of Software Test: Improving Practical Applicability Jan 2017).

Statement of Contribution

In all the included studies, the first author was the primary contributor to the research, approach and tool development, study design, data collection, analysis and reporting of the research work.

Other Studies

Papers authored or co-authored by Daniel Flemström but not included in this thesis:

1. SAGA Toolbox: Interactive Testing of Guarded Assertions. **Daniel Flemström**, Thomas Gustafsson, Avenir Kobetski. Published at The 10th IEEE International Conference on Software Testing, Verification and Validation (ICST'17).

2. A Research Roadmap for Test Design in Automated Integration Testing of Vehicular Systems. **Daniel Flemström**, Thomas Gustafsson, Avenir Kobetski, Daniel Sundmark. Published at The Second International Conference on Fundamentals and Advances in Software Systems Integration (FASSI'16).

3. Improving Introductory Programming Courses by Using a Simple Accelerated Graphics Library. Thomas Larsson, **Daniel Flemström**. Published at The Annual SIGRAD Conference; Special Theme: Computer Graphics in Healthcare (SIGRAD'07).

4. A Prototype Tool for Software Component Services in Embedded Real-time Systems. Frank Lüders, **Daniel Flemström**, Anders Wall and Ivica Crnkovic. Published at The International Symposium on Component-Based Software Engineering (CBSE'06).

5. Software Components Services for Embedded Real-Time. Frank Lüders, **Daniel Flemström**/ and Anders Wall. Published at The Working IEEE/IFIP Conference on Software Architecture (WICSA'05).

Contents

Part I

Thesis Summary

1 Introduction

Diligently controlling your dish washer, keeping you safe while you are sleeping in an airplane or traveling by train; embedded systems are part of our lives like never before. Embedded systems [3] typically consist of a computer that reads sensors such as buttons or pedals and interacts with hardware that controls physical entities such as breaks or engines, often with strict time constraints and without human interaction. To cope with size and complexity, they are developed in smaller modules that can be successively integrated, possibly at different levels of abstraction, to form the complete system. This *integration phase* [3] is one of the most critical phases in the development of embedded systems and includes a substantial amount of corresponding test effort.

In the software development process, the test and verification activities accounts for around half of the total development cost [5]. Up to 40% of this cost is the cost of integrating software modules to form a complete system. At the same time, several studies [2, 6, 12, 4] have found that a considerable amount of test effort is wasteful due to people, unknowingly or by necessity, performing similar (or overlapping) test activities. Consequently, test cases may end up to be similar, partially or wholly. In this thesis we focus on how to reduce the test and verification effort during this phase by exploiting knowledge of such similarities.

The main contributions of this thesis are:

(C1) *A systematic mapping study on reuse of test artifacts between different levels of integration to avoid test similarities.*

(C2) *A method for similarity detection in test cases, written in natural language.*

(C3) *Empirical evidence on the nature and characteristics of test case similarities in integration level test cases, written in natural language.*

(C4) *A method for reducing test effort by considering test case automation order together with similarity-based reuse.*

The contributions of this thesis can be divided into two categories. While the contribution in the first category aims to avoid similar test cases all together by reusing test artifacts between levels of integration, the contributions in the other category seek to reduce test effort by utilizing information on existing similarities.

The contribution (C1) represents the first category whereas the contribution (C2) is the base for the second category. More specific, the second category is represented by the entirety of contributions (C2)–(C4). Further details on these contributions are presented throughout the thesis.

This thesis is organized into two parts: “Thesis Summary” and “Papers”. In the first part, we present a high level summary of the research, whereas the specific details are presented in the published papers, included in the second part.

In the thesis summary, we first present an overview of the concepts and principles used in this thesis. The research questions and selected research methods are described in Section 3, followed by the contributions of this thesis that are discussed in more detail in Section 4. A summary of the published papers is included in

Section 5. We further present a selection of the most important related work in Section 6. The thesis summary is finally concluded in Section 7 where we present our conclusions and suggestions for future work.

2 Background

In this section we outline the central concepts and principles used in this thesis. We start with an overview of the purpose and concepts of software testing in Section 2.1. This overview takes the perspective of the widely accepted V–model [10] of software development. We continue with describing the applicable concepts and principles at one particular level of software testing: system integration testing in Section 2.2. In Section 2.3 we introduce similarities and test cases. Causes of test similarities are discussed in Section 2.4. Finally we conclude with an introduction to the test effort reduction concepts used in this thesis in Section 2.5.

2.1 Software Testing

Intuitively, the purpose of software testing is to make sure that the developed system works as expected [1]. The focus of this activity has shifted over time, from being a debugging aid, towards a structured approach for revealing bugs.

As of today, there exist many different testing approaches for different kinds of software systems [19]. In this thesis we focus on testing of *embedded* system software. Embedded systems typically interact with hardware that controls physical entities, such as actuators and sensors [3], as opposed to desktop applications such as Microsoft Word™, that mainly interacts with a user.

To define the scope of our work, we discuss how software testing fits into the software development process in its entirety. In a software development process, test design ideally starts in parallel with requirements engineering. The work flow of the development process is often described using the V-model [10]. Since its introduction in the 90s, the V-model has been widely accepted in the embedded software community, and it has been adopted in numerous variants [3, 15]. In this section we have chosen to present a simplified version of the V-model to illustrate the concepts and principles used in this thesis. Figure 1 illustrates the relevant concepts of that model. For a more extensive introduction to the V-model we refer the reader to the work of Forsberg et al. [10].

At the left side of the V-model, the system is broken down in a number of abstraction levels that correspond to an increasing level of detail in the specification. The top level specifies the complete product with both software and hardware, while the bottom level specifies the software implementation units. These units are successively integrated on the right side of the V-model to finally form the complete product.

Looking into more detail at the V-model, the functional specifications and their corresponding test specifications are developed iteratively at each level. The functional specifications are passed on to the next level of abstraction as soon as they have been approved. As the level of detail increases, more details are added to the

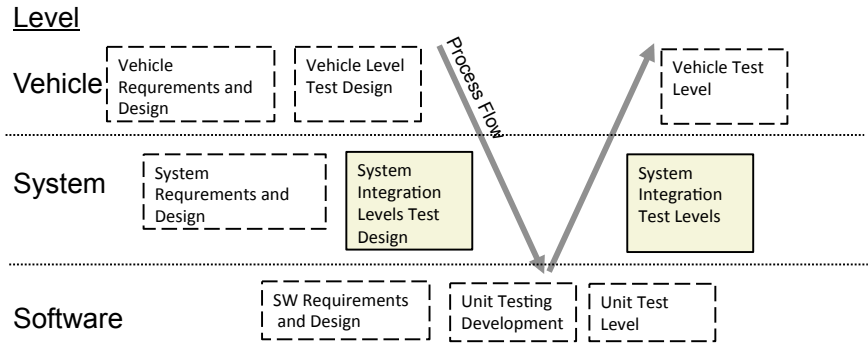


Figure 1: In the V-model, requirements and test specifications are created at the left side of the model and executed at the right side.

design. At the bottom of the V-model, the specifications have been divided into implementable units that can be tested in (more or less) isolation.

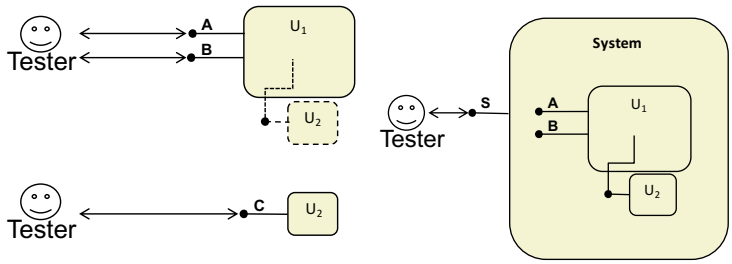
As soon as units are implemented and tested, the integration work begins. This is illustrated in Figure 1 by the right process flow arrow in the V-model. At the integration level(s), the software is integrated into larger subsystems until the integrated subsystems, together with the target hardware, form a complete product. As described earlier, a set of test cases with level-specific test objectives are created at each such level. Examples of such objectives are code coverage (at unit level) and interface compliance or timing accuracy at the system integration levels [19]. These differences are further discussed in Section 2.2.

2.2 System Integration Level Testing

As the main focus of this thesis is the system integration level, we devote this section to further discuss the system integration level(s). As previously discussed, the purpose of the system integration level(s) is to assemble all software units into a resulting, complete, software system. During this integration work, the main test objective is to make sure that the software works according to specification, both between the integrated parts and as an entirety. The latter, when the system is considered from a end users point of view, is also referred to as end-to-end testing [24].

Depending on the characteristics of the system under development, different number of *system integration levels* and *integration strategies* may be applied. Pezzè and Young [19] suggest an integration level where the test objective is based on software module compatibility, followed by a system level that specifically targets the end-to-end functionality. However, in this thesis we do not assume any particular number of integration levels. We simply note there are a number of integration levels and each level has its own level of abstraction and test objectives.

Pezzè and Young [19] describe different integration strategies, where the simplest strategy is the "no strategy" or big-bang integration. This means that no integration work is done until all software has been implemented. As a consequence all modules



(a) Unit Level Testing. (b) System Level Testing.

Figure 2: Testing on different levels of abstraction.

are integrated and tested at the same time. Other strategies such as "bottom up" or "top down" integration, promotes successive integration of the system in small increments with thorough testing at each increment. Such a successive approach requires that the parts of the system that have not yet been implemented must be stubbed or simulated. The difference between a stub and a simulated subsystem is that the former returns some default value for each request, regardless of the input values, while the latter returns some estimated value based on the input. Sometimes a combination of the bottom up and top down approach is required (Pezzè and Young define this as the sandwich approach in [19]).

Figure 2 illustrates the motivation behind a successive integration strategy. In particular it shows the difference between unit level test and system (integration) level test. In Figure 2a the tester can verify all the interfaces of the units U_1 and U_2 separately. Any faults are confined to either U_1 or U_2. We further note that, at the unit level, we may still perform the unit tests on U_1, even if unit U_2 is not implemented, by using a stub or simulation as discussed previously. This is illustrated by the dashed line around U_2 in Figure 2a.

Turning to the system integration level in Figure 2b, we see that the level of abstraction only allows access to the system interface (**S**). This interface allows testing of the end-to-end functionality only, since lower level interfaces such as $\mathbf{A} - \mathbf{C}$ may be hidden from the system interface (**S**). A fault in the end-to-end behavior may origin from *either* U_1 or U_2 *or* in the communication between them.

Connecting back to the discussion on integration strategies, the examples in Figure 2 illustrate the importance of not using the big-bang integration strategy. If we consider U_1 and U_2 in the same figure as sub systems instead of units, we see that this discussion can be generalized to any number of integration levels. In Section 2.4 we discuss the potential risk of redundant test effort with the increasing number of integration levels.

From the theoretical perspective of the V-model, we now turn to a more practical perspective: how an embedded system is tested in practice. Figure 3 describes a conceptual overview of an embedded system test environment adapted from the work of Broekman [3]. The most central part in the figure is the *SUT* which is an abbreviation for **S**ystem **U**nder **T**est. A SUT is typically the composite of several sub systems that have been integrated into one larger (sub)system. To allow efficient

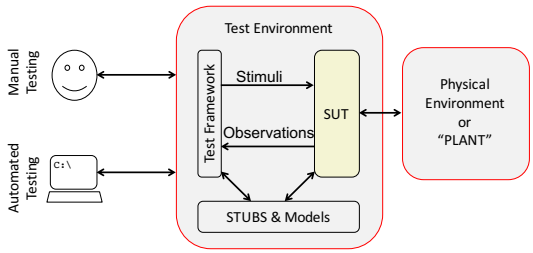


Figure 3: Conceptual overview over a System Under Test(SUT) driven by a test framework and interacting with a physical environment using stubs and simulation for the parts of the system that are not available.

integration strategies, missing sub systems are often implemented as *stubs*. The SUT interacts with the *physical environment* (sometime also called the "Plant"), consisting of actuators and sensors. This environment is typically realized partly by hardware and partly by real time simulation models. If the testing objective is functional verification, i.e. switching on a lamp in the cabin of a train, a purely simulated environment may be sufficient. On the other hand, if the objective of the test activity is to verify that the timing behavior is according to specification, a software simulated approach is often not adequate. In such cases the software needs to be executed on the target hardware, while most of the remaining environment may be realized as a real time simulation model.

The test environment in Figure 3 is controlled and managed by some kind of *Test Framework*. The framework provides means of managing and operating the system under test, either for *Manual Testing* or *Automated Testing*. Typical tasks for such a framework is to handle configuration data and provide an programmable interface to facilitate scripting stimuli to the SUT.

2.3 Test Case, Test Steps and Similarities

The terms test cases, test steps, and similarities are often used in this thesis. To better understand these terms, let us consider a concrete example shown in Table 1. The table illustrates a manual test case written in natural language, side by side with its automated counterpart[1]. The first column named *Step*, indicates the order of the different instructions so we can easily refer to them by number. The second column named *Stimuli* shows the instructions to the tester. These instructions can written be on a high level, such as the first step in the example, or it may be more detailed, as: push the button "X". For each such stimuli, there may also be an expected *reaction*, which tells the tester what he should expect from the system. Typically this means checking a certain signal level or the state of some indicator on a control panel. The rightmost column named *Automated Script* shows an example of how the stimuli could be automated, using a simplified automation language. When a test case has

[1]Due to space limitations, the following information is omitted from the table: Information on expected reaction and corresponding automated code, meta information such as sub system, responsible tester, etc.

Table 1: Obfuscated example of a test case, containing automated test steps.

Step	Stimuli	Automated Script
1	Initial State: Train is ready to drive	`train_init(READY)`
2	Run the train at speed 1234 km/h	`speed_set(1234)`
3	Set status to faulty for sensors XYZ for the motor car axle x	`fault_inject(XYZ,x)`
4	Set XYZ status to non faulty	`fault_clear(XYZ)`
5	Set slide XYZ status as active	`slide_status(XYZ,1)`
6	Set status to faulty for sensors XYZ for motor car axle x	`fault_inject(XYZ,x)`
7	Clean up....	`reset()`

been automated, it can be executed by a computer without manual intervention. Note that the script code for the expected reaction is also omitted in the example.

If we read the stimuli instructions carefully, we see that the step 3 and 6 have the same semantic meaning, but are expressed slightly differently. This is even more obvious in the rightmost column, where the automation code is identical for the two test steps. Finding similarities and making use of that information to reduce test effort is the core of this thesis.

2.4 Causes of Test Similarities

In this section we illustrate some causes of similarities by revisiting the V-model in Figure 1. We observe that the software is *i)* split into several sub systems and *ii)* tested at many different levels. For some less complex system components, the interface may remain unchanged throughout the integration levels, introducing the risk that similar, or even possibly redundant, test cases are written at each subsequent integration level. This is especially true when the integration levels correspond to different organizational units.

To make things even worse, there are often many versions and variants [6, 20] of the software, developed at the same time. This further increases the risk of ending up with test cases that are similar to each other [17].

As a practical example, we consider the case where the same subsystem is tested twice; first in a simulated environment, and later, in a hardware in the loop environment (at the same level of abstraction). In such a case, the instructions to the tester on "which lever to pull", and in which order, may be the same, but the instructions on what he should expect from the system are typically different (interface compliance versus timing behavior, for example). Yet another risk of introducing redundant effort, is the habit of adding extra test cases, "just to be on the safe side" [4].

The sheer number of test cases, sometimes counted in thousands [16], and the number of test engineers involved further adds to this risk of redundant test effort. Apart from the obvious disadvantage of writing and executing the same test instructions several times, time may also be wasted if there is an error in the original instruction, and these are copied into several other test cases. In the worst case, the

same error has to be discovered and corrected multiple times.

2.5 Thesis Scope for Test Effort Reduction

There exist many different approaches to reduce test effort [5, 23]. In this thesis we focus on methods that either avoid that similarities occur or methods that use the knowledge of similarities to reduce effort. One approach to avoid writing similar test cases, is to reuse test artifacts. Pérez and Kaiser [17] suggest reuse across different levels of integration. This is one direction of interest we have examined thoroughly in this thesis. We denote such reuse between different levels of integrations as *vertical reuse*.

Another known way of reducing *manual* test effort is to automate test cases. Automating a test case involves translating the manual instructions into some form of scripting counterpart that can be automatically executed by a computer, without human intervention. Table 1 shows an example of a test case written in natural language that has been automated.

In Section 3 we outline the research questions of this thesis, that will further add to the knowledge of test effort reduction using the knowledge of similarities.

3 Research Summary

In this section we present the research objective, followed by the research questions. We then discuss and motivate the choice of research methods used for answering the research questions.

3.1 Research Questions and Methodology

The voices of our industrial research partners are unanimous: we suspect that a great deal of our test effort is potentially redundant across software integration test levels. This particularly involves test cases ending up having similar contents. This phenomenon has also been reported sporadically in academia such as in [2, 6, 12].

Consequently, our research objective is to **discover the test effort reduction potential by using similarity knowledge within the context of integration testing in industrial embedded software development**. More specifically, we are interested in reducing test effort either by avoiding similarities or by using the knowledge of similarities.

In this thesis, we thus pose the following research questions:

RQ 1. *What evidence on vertical reuse as a method to avoid similarities in test artifacts for embedded systems exists in the academic discourse as of today?*

We denote reuse of any test artifact between different levels of integration as *vertical reuse*. Such artifacts may be models of the system, test cases or even test results. Since our industrial partners argue for vertical reuse as one way to avoid having similarities, the purpose of this question is to collect evidence on support for this claim in academia.

RQ 2. *To what extent can similarity information be used to reduce the test effort in industrial embedded system projects with manual test cases written in natural language?*

How and to what extent test effort can be reduced using information on similarities depends on the nature and characteristics of the similarities. The nature of the similarities describes if the similarities occur in the form of whole test case overlapping, or if test cases are similar only to its parts, the test steps. By characteristics, we refer to information such as: how often and where similarities exist; e.g., do similarities occur more often within the same test case, between levels, or even between development projects.

RQ 3. *What effect does similarity-based reuse and automation order have on test effort in projects with manual test cases written in natural language that are subject to automation?*

Automation in this context involves translation of the manual instructions, one by one, to an automatically executable script counterpart. Intuitively, if we can reuse the automation script of a test instruction that has already been automated, the automation effort would be reduced. Indirectly this means that the amount of manual testing could be reduced in a quicker pace. The purpose of this question is to gain a deeper understanding on how similarity-based reuse and automation order affects the test effort in general, and manual test execution effort in particular.

Common for all the research questions is that we put particular interest in manual system integration test specifications, written in natural language. In order to answer our research questions, we have undertaken a number of studies that have been published in three papers, denoted Paper A, Paper B and Paper C. The relation between these papers, the research questions, and the chosen research method is presented in Table 2.

Table 2: Research methods used and research questions answered in papers A–C.

Research Method	**Paper A**	**Paper B**	**Paper C**
Systematic Mapping Study	$RQ1$		
Industrial Case Study		$RQ2$	$RQ3$
Solution Formulation		$RQ2, RQ3$	$RQ3$

To answer the first research question we needed to identify the forums, authors and number of papers produced over time, as well as a classification of the different ideas presented in the reviewed papers. This together with the uncertainty of the existence of such work led us to the choice of systematic mapping study. This study was published in Paper A as shown in Table 2.

To answer the second question it was necessary to reveal whether similarities exist at all at the system integration level and, in the event that such similarities do

exist, between what test artifacts (e.g., are there more similarities within or between test cases, sub systems or even projects) they occur. This question required us to formulate a solution to determine the similarity between two test steps written in natural language. Further, since we argue for that the nature and characteristics of similarity occurrences will influence how we can use the information for test reduction purposes, we chose to perform an industrial case study. The resulting similarity method together with the results of the case study was published in Paper B and contributes in answering RQ2.

To answer the third question, a simulation of similarity-based reuse were applied to the process of automating test cases together with a prioritization technique. We further formulated a solution to the prioritization problem. To evaluate our method we chose to perform an industrial case study, where the automation process, using different automation ordering methods, as well as the testing process were simulated. The results were published in Paper C and contributes in answering RQ3.

The details of each research method and how they were used, are presented in the included papers.

4 Summary of Contributions

The top level research objective of this thesis is to discover the reduction test effort potential by using similarity knowledge in the context of integration testing in industrial embedded software development. To meet this objective, the research questions in Section 3.1 were further broken down and answered by the joint contributions of the publications included in this thesis.

In this section we present how the main contributions of the included papers contribute to answering the research questions and ultimately meeting the research objective of the thesis. The detailed contributions from each paper are presented in Section 5.

As discussed in the introduction, and illustrated in Figure 4, we categorize the contributions with respect to their approach to test effort reduction. The contribution in the first category, "avoiding similarities", makes use of general knowledge about where similarities usually occur to promote reuse. The primary goal is to *prevent* that similarities occur. The contributions in the second category, "using similarities", consider legacy sets of test cases for which we want to use the knowledge of similarities to *reduce existing* test effort.

The border between these two categories is however not exact. To illustrate this we consider the case where a large number of similar test cases are found. This information can be either be used for facilitating maintenance of similar, but not removable, test cases *or* be used for promoting test reuse.

The following research contributions are included in this thesis:

(C1) *A systematic mapping study on reuse of test artifacts between different levels of integration to avoid test similarities (Paper A).*
This scientific contribution is a collection of the existing research evidence on approaches to reuse test artifacts between levels of integration, *vertical*

test reuse, for embedded systems. The collected evidence includes the number and distribution of publications, methods, and motivations for reuse. This contribution mainly address RQ1 and the results have guided the research in Paper B and Paper C.

(C2) *A method for test similarity detection in test cases, written in natural language (Paper B).*
This contribution is an achievable, yet industrially scalable, method to detect similarities in manual test cases that are written in natural language. Further, by using this method, a large number of test cases can be encoded into an efficient representation that can be used with standard numerical algorithms. This method lays the ground work of collecting data for answering RQ2 and RQ3. The method is also used in Paper C to track similarities in manual test cases.

(C3) *Empirical evidence on the nature and characteristics of test case similarities in integration level test cases, written in natural language (Paper B).*
This contribution consists of empirical evidence on similarities from four industrial sized embedded software projects in the vehicular domain. It shows to what extent similarities occur in the set of test cases and where they occur. This knowledge contributed to RQ2 and guided the design of the effort reduction technique presented in Paper C.

(C4) *A method for reducing test effort by considering test case automation order together with similarity-based reuse (Paper C).*
This method uses the knowledge of unavoidable test similarities, to reduce test effort in general and manual execution test effort in particular. The reduction is accomplished by selecting an automation order that gives early and high manual effort reduction. The knowledge from this contribution is presented in paper C and contributes to RQ3.

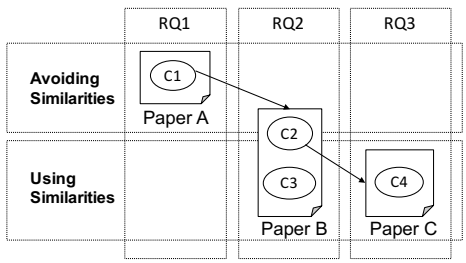


Figure 4: The contributions (C1–C4) from papers A–C answering the research questions.

Figure 4 illustrates how the contributions from the included papers contribute to each research question (RQ1–3 in Figure 4). The arrows show that Paper A influenced the research question RQ2 as well as the research conducted in Paper B. The arrow between C2 in Paper B and Paper C denotes the reuse of the method to identify similarities.

5 Summary of Included Papers

In this section we present an overview of the included papers. For each paper we give a short description of the paper followed by the paper's research objective, the summary of the method is presented and finally, we conclude with the results.

Paper A: Vertical Test Reuse for Embedded Systems: A Systematic Mapping Study

The paper [9] presents a mapping study on vertical reuse. The term *vertical test reuse* here refers to reuse of test cases or other test artifacts over different integration levels in the software development process.
Research Objective: The objective of the study was to provide an overview of the state of the art in the field of vertical test reuse for embedded system development.
Methodology: The study follows the systematic mapping study guidelines as proposed by Petersen et al. [18]. Following this guide, a set of research questions were formulated to meet the objective. A search strategy, constituting target data bases and relevant search terms, was developed and used for retrieving a base set of relevant papers. This set of papers were iteratively reduced using a set of inclusion/exclusion criteria. These criteria were guided by the research questions and the objective of the study. Lastly the papers were analyzed using a data extraction guide that was developed using the research questions.
Results: The study identified 11 primary studies on vertical test reuse for embedded systems, published in the period of 2002 to 2013. Most published work (8 out of 11 included studies) focus on solution proposals, but three studies target evaluation of vertical test reuse.

The primary motivation for vertical test reuse seems to be reduction of test effort by means of avoiding test rework over test levels. Other motivations include reduction of test case complexity, increased quality of test cases, and less costly test environment creation. The primary artifact being reused between test levels is abstract test cases or test models, and reuse is achieved through using adapters, formal methods or clone mining.

A majority of the primary studies use proof-of-concept as a means to evaluate their approaches on either down-scaled real or industrial-scaled examples. Few comparative evaluations exist in the field, with the exception of two primary studies that use industrially-scaled examples with sound comparative evaluation.

Paper B: Exploring Test Overlap in System Integration: An Industrial Case Study

The paper [7] presents an industrial case study on four embedded system development projects at the system integration level, conducted at Bombardier Transportation Sweden AB in Västerås.
Research Objective: The research objectives were to investigate to what extent there exist similarities[2] within testing at system integration level testing, and to

[2]In the published study we used the term test *overlaps* to conform to common terminology.

explore how these similarities are distributed over different test levels, projects and subsystems.
Methodology: The case study required a similarity function for the data collection. The challenges of identifying similarities were identified together with the industry. Considering the industrial context and state of art and practice, a similarity function was developed and evaluated using a subset of the available test cases. Since we had access to the full set of test cases for the four studied projects, the solution could be iterated internally instead of between the industrial partners and the academia until it was sufficiently mature to be used in the full scale industrial case study.

Following the guidelines by Runeson et al. [21] the exploratory industrial case study started with selecting Bombardier Transportation Sweden AB as the case organization and four projects of different sizes and complexity as units of analysis. A set of research questions were formulated to meet the objective. These questions, together with an on-site observation, guided the design of a data collection and analysis procedure. In a pre-analysis phase, features to be extracted (i.e., test steps) from the selected projects, together with criteria and a plan for evaluating the result were established. This phase was followed by an execution phase, where the developed similarity function was applied to the the extracted test steps of the four projects. In the data collection phase, data on found similarities were extracted. The collected data were finally analyzed with respect to the formulated research questions.
Results: Substantial test similarities were found within the system integration test levels. Most of these were confined to certain steps within test case sequences, and were due to reuse between projects. However, we were also able to find test similarities within projects, primarily in the form of similar preparation steps of test cases. Concerning test effort reduction potential, we could identify candidates for automation and simplified maintenance, but no candidates for removal (at least not without additional detailed analysis).

Paper C: Similarity-Based Prioritization of Test Case Automation

The paper [8] presents a test case automation strategy to minimize the manual test effort during the process of automating a set of test cases. The strategy is evaluated in a large industrial case study at Bombardier Transportation Sweden AB in Västerås.
Research Objective: The aim was to collect empirical evidence on the potential for reducing test effort by ordering, or prioritizing, the test cases for automation, given that we can reuse already automated parts of test cases. The objective of the case study was to evaluate, how such similarity-based reuse of automated test steps and prioritization of the test cases for automation, affects test effort.
Methodology: The problem of prioritizing test case for automation was identified together with our case organization, Bombardier Transportation Sweden AB. From the context of state of art and practice, a proposed solution was developed and evaluated using a subset of the available test cases. Since we had access to the full set of test cases for the four studied projects at Bombardier Transportation Sweden AB,

However during subsequent research, we have found that the term *similarities* is more suitable.

the solution could be iterated internally instead of between the industrial partner and the academia until it was sufficiently mature to be used at full scale. The last step of the technology transfer model is to compare the proposed solution against other available solutions. This was done in an industrial case study, following the guidelines of Runeson et al. [21]. For the study, Bombardier Transportation Sweden AB was chosen as the case organization and four embedded system projects of different size and complexity were selected as units of study. A set of research questions was formulated to i) see the extent of the effect and ii) compare different automation strategies. Manual test execution effort and execution time were chosen as dependent variables. Further, seven experiments with different prioritization approaches and configurations were defined. In the execution phase the defined experiments were executed on the four projects, and finally, the data from the experiment executions was collected and analyzed.

Results: The results show that for the projects considered, the expected manual test effort is in average 14 percentage points lower when using similarity-based reuse. Further, the best-performing automation order can reduce the expected manual test effort with 6 additional percentage points. The results also show that our proposed prioritization method is nearly as good as more resource demanding meta-heuristic approaches at a fraction of the computational time.

6 Related Work

To our knowledge there is one existing mapping study [5] on reducing test effort specifically by "reuse". In this study, 144 papers were examined and five different ways of reducing test effort were identified. Automation (50% of the papers) was one of the two largest groups of published papers. While the identified automation approaches mainly target effort reduction by *generating* test cases from either code at unit level, models or requirements, our focus is on reducing effort by e.g. automating a set of manual test cases at the integration level considering similarities.

In another literature study [23], Tiwari and Goel identify the state of the art on methods available for reducing test effort. The methods mainly address the situation where test cases are designed from start to be reusable, by e.g., utilizing model based approaches or templates, while our approach addresses the situation where a large manual test set already exists.

The phenomenon of having similar systems to test is acknowledged in the work of Runesson and Engström [20]. The authors define a 3D process model where possible reuse of test cases may occur between different *versions* or *variants* of the system but also *between levels of integration.* When performing software testing across these dimensions, it is not uncommon that a certain amount of the testing activities are repeated throughout the software development cycle. Engström and Runesson have investigated this in an industrial case study [6] where several cases of such redundant testing work were identified. They further conclude that tool support is required in order to avoid, or efficiently handle, such situations. Although the authors focus on software product lines and regression testing, the definitions of the dimensions, the phenomenon of test overlap and the need for tool support still applies to our

problem domain. One way to avoid these overlaps is to conduct some form of reuse of test cases across the dimensions defined in [20]. Focusing on test reuse between levels of integration in embedded systems, we have identified three approaches of reuse approaches in a recent mapping study [9]. Most of these approaches (creating adapters for different levels, or using formal abstractions) targets design of test cases rather than existing test cases. One exception is the "clone mining" approach in [2]. The authors define different types of test clones and use a data mining tool to find them and address the problem of similarities (referred to as *clones* in their paper) by different transformations. The focus of the paper is automatic test code at unit level, but on a conceptual level, it has influenced the direction of our work to focus on similarities in manual test cases written in natural language.

There is an overwhelming amount of approaches for using similarity as a means to reduce test effort. Hemmati et. al [14], investigated 320 different similarity-based selection techniques to help select test cases. These methods are based on similarity measures that returns a similarity score for a set of test cases or a full test case. The goal of their study was to minimize the pairwise similarity and get a reduced test set, while our approach seek to identify overlaps that cannot be eliminated and handle them in an appropriate way. Further, many of the information retrieval approaches are computationally intensive and may take up to several hours or even days to complete on a large set of test cases. The approach we suggest should, on the other hand, be possible to include in an online tool that gives feedback within minutes or even seconds. A recent doctoral dissertation [13] focused entirely on reducing system testing effort using knowledge of similarities.

To make practical use of identified similarities in large sets of test cases, there is a need for tool support [6, 16] such as a recommendation system. Gasparic et al. conducted a literature study [11] on recommendation systems but found that there are few recommendation systems that operate outside the source code domain.

When deciding point in time and which test case to automate first, there are several factors to consider. The work that is most similar to ours, is a prioritization approach by Sabev and Grigorova [22]. They base their selection model on the quotient between manual and automated test effort. This model promotes automation of test cases that has a high manual effort and a low automation effort. The effort estimation effort builds on a set of, by the authors, identified factors, such as "large data inputs" and intrinsic information about the test cases. Our goal has been to avoid any kind of factors that cannot be determined automatically from the test case itself. The main difference to our work is that we also take into account the future gain of automation, given the set of already automated test cases and the set of remaining test cases.

7 Conclusions and Future Work

We have explored the reduction potential of similarity knowledge within the context of integration testing in industrial embedded software projects. The main contributions of this thesis include a review of the state of the art on test reuse between levels of integration. We have also developed an efficient method to find and encode

similarities in test cases, written in natural language. Using the method we have gathered empirical evidence on the occurrence of similarities in test cases and developed a prioritization algorithm for test automation order. By using the order, the manual test execution effort is reduced as quick and as early as possible.

We conclude that avoiding similarities by reusing test artifacts between levels of integration, practically requires changes in the test case design due to the different interfaces at the different levels of integration. Our studies further show that there is a substantial amount of similarities in today's test cases and that these similarities tend to be limited to *partial* test cases. This confirms the need of a tool that e.g. facilitates propagation of updates to keep similar test steps consistent. Another example of such tool functionality is to provide code completion for test case automation, using knowledge of similar test steps that have been automated before (i.e., *similarity-based reuse*). Such similarity-based reuse contributes to a substantial reduction of the test case automation effort. Finally we conclude that when considering similarity-based reuse, the order in which test cases are automated has a considerable impact on the manual test execution effort reduction pace.

While there are many factors influencing the decision on what to automate and when, we have only considered the automation effort and future gain based on number of steps and similarities in the test cases. We are, however, aware of that there are other important factors to consider. Increasing the number of factors considered in the prioritizaion of the automation order is an interesting future research topic. One objective of particular interest is how take into account an even workload distribution for the test employees that are dividing their time between the automation work and the test execution work.

Moreover, the current similarity function is very strict, but it has proven to work fairly well in the case studies since many of the test steps were nearly identical. A more relaxed similarity function would possibly find more similarities in the general case, but may be more time-consuming and also report more non-similar test steps as similar. Thus there is a need to investigate the gain and penalty for introducing different standard similarity functions with respect to similarities found, false positives and execution time to see how much the test effort reduction rates improves.

Finally, in the current approach, only the number of test steps are considered in cost model of the prioritization model. We argue that there is a need to investigate how the known factors, that can be automatically retrieved, can contribute to a more accurate model. Ultimately, we would like to know whether such model additions improve effort reduction due to the prioritization or not.

Bibliography

[1] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. (2008) (cit. on p. 4).

[2] Suriya Priya R. Asaithambi and Stan Jarzabek. "Towards Test Case Reuse: A Study of Redundancies in Android Platform Test Libraries". In: *Safe and Secure Software Reuse*. Ed. by John Favaro and Maurizio Morisio. Vol. 7925. Lecture Notes in Computer Science. 2013, pp. 49–64 (cit. on pp. 3, 9, 16).

[3] Bart Broekman. *Testing Embedded Software*. (2003) (cit. on pp. 3, 4, 6).

[4] Christof Ebert and Capers Jones. "Embedded Software: Facts, Figures, and Future". In: *Computer* 42.4 (2009) (cit. on pp. 3, 8).

[5] Frank Elberzhager, Alla Rosbach, Jürgen Münch, and Robert Eschbach. "Reducing Test Effort: A Systematic Mapping Study on Existing Approaches". In: *Information and Software Technology* 54.10 (2012), pp. 1092–1106 (cit. on pp. 3, 9, 15).

[6] Emelie Engström and Per Runeson. "Test Overlay in an Emerging Software Product Line – An Industrial Case Study". In: *Information and Software Technology* 55.3 (2013), pp. 581–594 (cit. on pp. 3, 8, 9, 15, 16).

[7] Daniel Flemström, Wasif Afzal, and Daniel Sundmark. "Exploring Test Overlap in System Integration: An Industrial Case Study". In: *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA'16)*. (2016), pp. 303–312 (cit. on p. 13).

[8] Daniel Flemström, Pasqualina Potena, Wasif Afzal Daniel Sundmark, and Markus Bohlin. ""Similarity-Based Prioritization of Test Case Automation"". In: *Special Issue of Software Quality Journal on Automation of Software Test: Improving Practical Applicability* (sumbitted) (cit. on p. 14).

[9] Daniel Flemström, Daniel Sundmark, and Wasif Afzal. "Vertical Test Reuse for Embedded Systems: A Systematic Mapping Study". In: *2015 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA'15)*. (2015), pp. 317–324 (cit. on pp. 13, 16).

[10] Kevin Forsberg and Harold Mooz. "The Relationship of System Engineering to the Project Cycle". In: *INCOSE International Symposium*. Vol. 1. 1. (1991), pp. 57–65 (cit. on p. 4).

[11] Marko Gasparic and Andrea Janes. “What Recommendation Systems for Software Engineering Recommend: A systematic literature review”. In: *Journal of Systems and Software* 113 (2016), pp. 101–113 (cit. on p. 16).

[12] Michaela Greiler, Arie van Deursen, and Andy Zaidman. “Measuring Test Case Similarity to Support Test Suite Understanding”. In: *Proceedings of the 50th International Conference on Objects, Models, Components, Patterns (TOOLS'12)*. (2012) (cit. on pp. 3, 9).

[13] Benedikt Hauptmann. “Reducing System Testing Effort by Focusing on Commonalities in Test Procedures”. PhD thesis. Technische Universität München, 2016 (cit. on p. 16).

[14] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. “Achieving Scalable Model-Based Testing Through Test Case Diversity”. In: *ACM Transactions on Software Engineering Methodology* 22.1 (2013), 6:1–6:42 (cit. on p. 16).

[15] Oliver Hoehne. “The SoS-VEE Model: Mastering the Socio-Technical Aspects and Complexity of Systems of Systems Engineering (SoSE)”. In: *INCOSE International Symposium*. Vol. 26. 1. (2016), pp. 1494–1508 (cit. on p. 4).

[16] Remo Lachmann and Ina Schaefer. “Towards Efficient and Effective Testing in Automotive Software Development.” In: *GI-Jahrestagung*. (2014), pp. 2181–2192 (cit. on pp. 8, 16).

[17] Abel Marrero Pérez and Stefan Kaiser. “Integrating Test Levels for Embedded Systems”. In: *Testing: Academic and Industrial Conference-Practice and Research Techniques (TAIC PART'09)*. (2009), pp. 184–193 (cit. on pp. 8, 9).

[18] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. “Systematic Mapping Studies in Software Engineering”. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08)*. Swinton, UK, (2008) (cit. on p. 13).

[19] Mauro Pezzè and Michal Young. *Software Testing and Analysis: Process, Principles and Techniques*. (2008) (cit. on pp. 4–6).

[20] Per Runeson and Emelie Engstrom. “Software Product Line Testing–A 3D Regression Testing Problem”. In: *Fifth International Conference on Software Testing, Verification and Validation (ICST'12)*. (2012), pp. 742–746 (cit. on pp. 8, 15, 16).

[21] Per Runeson and Martin Höst. “Guidelines for Conducting and Reporting Case Study Research in Software Engineering”. In: *Empirical Software Engineering* 14.2 (2009), pp. 131–164 (cit. on pp. 14, 15).

[22] Peter Sabev and Katalina Grigorova. “Manual to Automated Testing: An Effort-Based Approach for Determining the Priority of Software Test Automation”. In: *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering* 9.12 (2015), pp. 2123–2129 (cit. on p. 16).

[23] Rajeev Tiwari and Noopur Goel. “Reuse: Reducing Test Effort”. In: *SIGSOFT Software Engineering Notes* 38.2 (2013), pp. 1–11 (cit. on pp. 9, 15).

[24] Wei-Tek Tsai, Xiaoying Bai, Richard Paul, Weiguang Shao, and Vijyant Agarwal. “End-to-end Integration Testing Design”. In: *25th Annual International Computer Software and Applications Conference (COMPSAC'01)*. (2001), pp. 166–171 (cit. on p. 5).