Doctoral Thesis

Test optimization for Highly-Configurable Cyber-Physical Systems

Author:
Urtzi Markiegi Gonzalez

Supervisors:
Dr. Leire Etxeberria Elorza
Dr. Goiuria Sagardui Mendieta

PhD Program in Applied Engineering
Computer and Electronics Departament
Faculty of Engineering
Mondragon Unibertsitatea

Arrasate-Mondragon
May 2021

Gizonen lana jakintza dugu; ezagutuz aldatzea,
naturarekin bat izan eta harremanetan sartzea.
Eta indarrak ongi errotuz gure sustraiak lurrari lotuz,
bertatikan irautea: ezaren gudaz baietza sortuz,
ukazioa legetzat hartuz beti aurrera joatea.

El trabajo humano es conocimiento: conocer y transformar,
Hermanarse con la naturaleza y llegar a desvelarla
Crear de la negación lo positivo,
Y tomando la contradicción por ley continuar avanzando.

Human work is knowledge: knowing and transforming,
To become twinned with nature and to unveil it.
To create from negation the positive,
And taking contradiction as law, to continue advancing.

(Xabier Lete)

Nekane, Irati eta Nagore.
Aita, Ama, Anaiz eta Amona.

Nigan sinistu eta beti bezala
zuen babesa eman didazutelako.
Urtzi

Acknowledgments

Partially putting aside teaching and (what I thought was) research to focus on this thesis has been a challenge I had been waiting for for years. They have been five long years, with good moments but also some more difficult ones. But if one thing is clear to me, it is that if I have reached the end, it has been thanks to the hard work and to the unconditional support of the great team-family.

First of all I would like to thank Leire and Goiuria for their constant support, patience and thoroughness during all this journey, otherwise this thesis would not have happened. No, I did not know how to do research. So, thank you for teaching me to question everything and to improve the technique for experimentation. Thank you for setting milestones and giving me room to adapt and also for the rigorous feedback. Thank you for trusting me.

Aitor, this thesis continues the research work you started. I am very privileged to have your support in my research, both technically and in writing papers. I have learned a lot. You are teaching me day after day. Thanks for your patience, generosity and the time you have dedicated to me. But thanks also for the good times and encouragement you have given me.

But this work would not have been possible without the collaboration of the rest of my colleagues, who have made an effort to cover for me when I was not there: Osane (in 5 years with SMC and much more), Arkauz and Cuenca (with the Mercedes project), Joseba (in the Master), Alain (in multiple *opportunity* projects), Iñigo and Xabi V. (in our European adventures) and Gorka (replacing me in databases). In addition, there are not few who have kept the flame of illusion and motivation with their advice and encouragement or simply listening to me: Osane, Felix, Miren, Garitano, Aitzol thanks for your support. I would also like to thank Roberto, Sagarna and Nekane for your close and unconditional support and backup.

The initial stage of the thesis made me go back to my student days and make new *friends*. Thank you Fernando, David, Unai, Dani for the shared good times. Also, many thanks to the lecturers at that stage (Artetxe, Dani, Paula, etc.). Your training has been fundamental for this work. Thanks Susanne, for helping me in this learning process also with language issues and for your support.

And a new adventure arrived, in due time, but before I had finished the thesis. With the coordination of the Computer Science degree, I want to thank Txema for his patience and generosity and Patxi, Jone and Miren for their help all this time. And I do not forget my colleagues of the title team (Enaitz, Mikel, Jon, etc.), you have suffered my absence and my enthusiasm. I do not know which is worse, but we will have time to improve it.

Soon others will finish: Dani, Velez, come on, there's nothing left! Miriam, Jon, I am sure you will do a brilliant job!

Mondragon Goi Eskola Politeknikoa, Eskola, is my second home. And I want to thank all the colleagues of Eskola for giving me the opportunity to contribute with this thesis.

Finally my family. Nekane, Irati, Nagore, Aita, Ama, Anaiz and Amona. I love you. I owe you many good moments, and sorry for having paid you back with bad ones. You are always there, helping, listening and putting up with me. As *amona Maritxu* says, you are "My refuge".

Thank you all, it's awesome to feel the support of such a big family.

Eskerrik asko!

Declaration

Hereby I declare that this document is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Urtzi Markiegi Gonzalez
Arrasate-Mondragon, December 2021

This page intentionally left blank.

Abstract

Over the past decade, Cyber-Physical Systems (CPSs) have gained prominence as core-enabling technologies in the development of multiple domains, thanks to its ability to integrate digital capabilities with physical processes. Furthermore, the demand for configurability of CPSs has been increasing rapidly to respond to changing business requirements.

When engineers approach the development of Highly-Configurable Cyber-Physical Systems (HCCPSs), product line engineering techniques are often adopted, taking advantage of variability management strategies that allow handling a large number of configurations. In addition, to address intrinsic challenges of testing CPSs, engineers are relying on simulation-based techniques, thus avoiding the need for building real prototypes and enabling testing at early stages. However, HCCPS testing is still a time-consuming challenge primarily due to the intensive resource consumption when simulating physical processes. Consequently, optimization of testing HCCPSs is paramount.

Several approaches have tackled the test optimization challenge, most of them focused on reducing the number of products to be tested, by selecting a representative subset. Other approaches have proposed optimization in terms of test case selection and prioritization. However, little attention has been paid to optimization of both, products and test cases, in a combined manner. This thesis aims at advancing the current practice of optimizing HCCPS testing by proposing a method to increase the fault detection rate in time-constrained scenarios. To this end, we propose a dynamic test prioritization approach combining both, products and test cases. The approach sets a test plan that executes small groups of test cases with products in iterative executions. After every iteration, the test plan is dynamically re-ordered, leveraging information of test cases being executed in specific products. The approach has been evaluated and validated for the specific context of HCCPSs, however, it might eventually pave the way for its use in other type of product lines.

Laburpena

Azken hamarkadan zehar sistema ziber-fisikoek (ingelesezko sigletako CPS) protagonismoa irabazi dute domeinu anitzen garapenean, gaitasun digitalak sistemen prozesu fisikoekin integratzeko duten trebetasunari esker. Gainera, CPSen konfiguragarritasun-eskaria azkar handitu da, enpresa-eskakizun aldakorrei erantzuteko.

Ingeniariek oso konfiguragarriak diren sistema ziber-fisikoen garapena lantzen dutenean (ingelesezko sigletako HCCPS), ohikoa da produktu-lerroetan oinarritutako ingeniaritza-teknikak hartzea, konfigurazio ugari erabiltzea ahalbidetzen duten aldakortasuna kudeatzeko estrategiak baliatuz. Gainera, CPSen proben berezko erronkei aurre egiteko, ingeniariek simulazioan oinarritutako tekniketara jotzen dute, prototipo errealak eraikitzeko beharra saihestuz eta etapa goiztiarretan probak egitea ahalbidetuz. Hala ere, HCCPSak probatzea denbora asko eskatzen duen erronka izaten jarraitzen du, batez ere prozesu fisikoak simulatzeko behar diren baliabideen kontsumo handia dela eta. Ondorioz, HCCPSen probak optimizatzea funtsezkoa da.

Hainbat ikuspegik heldu diote probak optimizatzeko erronkari; horietako gehienak probatu beharreko produktuen kopurua murriztean jarri dute arreta, azpimultzo adierazgarri bat hautatuz. Beste ikuspegi batzuek proba-kasuak hautatzeko eta lehenesteko optimizazioa proposatu dute. Hala ere, arreta gutxi jarri da biak, produktuak eta proba-kasuak, modu konbinatuan optimizatzeko. Tesi honen helburua HCCPSen probak optimizatzeko egungo praktika aurreratzea da, denbora mugatua duten egoeratan akatsen detekziotasa handitzeko metodo bat proposatuz. Horretarako, probak lehenesteko ikuspuntu dinamiko bat proposatzen dugu, produktuak eta proba-kasuak konbinatzen dituena. Lehenik froga-plan bat ezartzen da, eta froga-kasuen multzo txikiak exekutatzen dira produktuekin batera iterazio ezberdinetan. Iterazio bakoitzaren ostean, proba-plana dinamikoki berrantolatzen da, produktu es-

pezifikoetan exekutatzen diren proba-kasuen informazioaz baliatuz. Ikuspegia HCCPSen testuinguru espezifikorako ebaluatu eta balioztatu da, baina beste produktu-lerro batzuetan erabiltzeko bidea erraztu dezake.

Resumen

A lo largo de la última década, los sistemas ciber-físicos (CPS de sus siglas en inglés) han ganado protagonismo como tecnologías esenciales en el desarrollo de múltiples dominios, gracias a su habilidad para integrar las capacidades digitales con los procesos físicos de los sistemas. Además, la demanda de configurabilidad de los CPS ha aumentado rápidamente para responder a los cambiantes requisitos empresariales.

Cuando los ingenieros abordan el desarrollo de sistemas ciber-físicos altamente configurables (HCCPS de sus siglas en inglés), es habitual que adopten técnicas de ingeniería basadas en líneas de productos, aprovechando las estrategias de gestión de la variabilidad que permiten manejar un gran número de configuraciones. Además, para hacer frente a los retos intrínsecos de las pruebas de los CPS, los ingenieros recurren a técnicas basadas en la simulación, evitando así la necesidad de construir prototipos reales y permitiendo la realización de pruebas en etapas tempranas. Sin embargo, probar los HCCPS sigue siendo un reto que requiere mucho tiempo, principalmente debido al intenso consumo de recursos necesario para la simulación de los procesos físicos. En consecuencia, la optimización de las pruebas de los HCCPS es primordial.

Varios enfoques han abordado el reto de la optimización de las pruebas, la mayoría de ellos centrados en la reducción del número de productos a probar, mediante la selección de un subconjunto representativo. Otros enfoques han propuesto la optimización en términos de selección y priorización de casos de prueba. Sin embargo, se ha prestado poca atención a la optimización de ambos, productos y casos de prueba, de forma combinada. Esta tesis tiene como objetivo avanzar la práctica actual de optimización de las pruebas de los HCCPS proponiendo un método para aumentar la tasa de detección de fallos en escenarios con de tiempo limitado. Para ello, proponemos un enfoque dinámico de priorización de pruebas que combina tanto productos como casos de prueba. El enfoque establece un plan de pruebas que ejecuta pequeños

grupos de casos de prueba con productos en ejecuciones iterativas. Después de cada iteración, el plan de pruebas se reordena dinámicamente, aprovechando la información de los casos de prueba que se ejecutan en productos específicos. El enfoque ha sido evaluado y validado para el contexto específico de los HCCPS, pero podría permitir allanar el camino para su uso en otro tipo de líneas de productos.

Contents

List of Figures

List of Tables

Acronyms

AE	Applictaion Engieering
API	Application Program Interface
AS	All-Signals Similarity of pairs
AT	Application Testing
AVM	Alternating Variable Method
BFS	Brute Force Strategy
CC	Condition Coverage
CIT	Combinatorial Interaction Testing
CPS	Cyber-Physical System
CRS	Commonality and Reuse Strategy
CW	Car Windows
DC	Decision Coverage
DE	Domain Engieering
DT	Domain Testing
DTO	Delete Transition Operator
FDC	Fault Detection Capability
FM	Feature Model
FODA	Feature-Oriented Domain Analysis
GA	Genetic Algorithm
GS	Grouped-Signals Similarity of pairs
HCCPS	Highly-Configurable Cyber-Physical System
HCS	Highly-Configurable System

HIL Hardware-in-the-loop

IT Industrial Tanks

LOR Logical Operator Replacement

MC/DC Modified Condition/Decision Coverage

MIL Model-in-the-loop

PAS Pure Application Strategy

PIL Processor-in-the-loop

PLE Product Line Engineering

PLSC Product-Line Structural Coverage

PSC Product Structural Coverage

RS Random Search

RWGA Random-Weighted Genetic Algorithm

SAS Sample Application Strategy

SBO Search-Based Optimization

SBSE Search-Based Software Engineering

SC Structural Coverage

SDLC Software Development Life-Cycle

SIL Software-in-the-loop

SLDV Simulink Design Verifier

SPL Software Product Line

SPLE Software Product Line Engineering

SUT System Under Test

TCAF Test Case Appearence Frequency

TET Test Execution Time

UAV Unmanned Aerial Vehicle

VCO Variable Change Operator

WBGA Weight-Based Genetic Algorithm

(1+1)EA (1 + 1) Evolutionary Algorithm

Part I

Foundation and Context

Chapter 1

Introduction

Contents

This chapter introduces the motivation and scope of the research work in this thesis. The research methodology is described. The main technical contributions are presented and finally, publications and related activities are summarized.

1.1 Motivation and Scope of the Research

Over the past decade, **Cyber-Physical Systems (CPSs)** have gained prominence as core-enabling technologies in the development of multiple domains. This is exemplified in manufacturing, where CPSs are considered to be one of the foundations of the so-called Industry 4.0, the latest industrial revolution [HWJ13, LBK15, Jaz14, Kim17, XXL18]. Likewise, CPSs expansion is also becoming a key technology in multiple domains such as healthcare, energy, infrastructures, military or transportation [SL12, Che17a, AMAAA17, SRJ+17]. Moreover, academic studies [PII+19, GBT21, YSN+20, NMP20] point to more complex and demanding systems in terms of intelligence, adaptability, reliability and security as the near future for CPSs.

According to the definition provided by Lee and Seshia, "a CPS is an integration of computation with physical processes whose behavior is defined by both cyber and physical parts of the system" [LS17]. For the development of CPSs, multiple approaches have been defined, adapted and applied [DLV12, Che17b, DH18], due to its complex and multidisciplinary nature. However, Krüger et al. highlighted that existing CPS development approaches share a common challenge: **variability** [KNK+17]. In addition, unlike traditional systems where variability is defined on hardware and software levels, variability definition on CPSs must consider component, context, hierarchy, quality and time aspects [KNK+17]. For example, these aspects describe reusable variable hardware (i.e., variability in components), the interrelationships and dependencies between components (i.e., hierarchy of variability), variants that must adapt their functionality to the context (i.e., variability in context), or quality criteria that must be adapted according to the scenario (i.e., variability in quality). We refer to **Highly-Configurable Cyber-Physical Systems (HCCPSs)** when the configurable CPS system must satisfy challenges raised by the domain in multiple of the described variable aspects. Thus, HCCPSs allow the generation of thousands or even millions of different variants from one single highly configurable system. Therefore, managing all these aspects of variability makes HCCPS development and validation challenging.

Often, variability management of highly-configurable systems is handled by adopting techniques from **Product Line Engineering (PLE)** [CDS07, San16, SKT+16]. PLE fundamentals allow the development of both the common and the variable parts of all products of a family at the Domain level. On the other hand, the development of each particular product is carried out at the Application level, specifying the use of variable parts [PBL05].

As we have moved into the 21st century, ensuring the right level of quality of systems is essential for the progress of our civilization [AO16]. A growing number of activities of our modern society depend on the proper functioning of systems. In other cases, specific levels of quality in terms of safety and security are a prerequisite for the operation of systems. Among the different verification and validation techniques that can be used to ensure these levels of quality, **testing** is considered to be one of the most prevalent techniques. However, the complex and uncertain nature of cyber-physical systems makes traditional validation methods expensive, time consuming or infeasible [BNSB16]. **Testing CPSs** must take into account the physical processes and their interaction with the software and the environment, which makes the testing activity more demanding. **Simulation models** provide an effective way to design and test CPSs by modeling the hardware and software processes that define the behavior of the system and context. These simulation models allow the execution of test suites, the selection of most suitable cost-effective test cases and the automated evaluation of test executions [BNSB16].

In the particular field of **testing, PLE** also differentiates the activities at the two levels. At Domain level the Domain Testing activity is held where defects in domain artifacts are uncovered and where reusable test artifacts for application testing are created. On the other hand, at Application level the Application Testing activity is conducted to detect defects by reusing domain artifacts. Within the various challenges posed by product line testing activities [ER11], the need for reduction of the large number of products that potentially need to be tested has been one of the most explored fields [dCMMCDA14, LHFRE15]. Since testing all possible combinations is unfeasible, most of the research works have focused on selecting [OMR10, POS+12] and prioritizing [SSRC14a, AHTL+16] a representative subset of products (at Domain level) to be tested.

Facing the challenge of testing HCCPSs, the aforementioned challenges of both testing CPSs and testing of PLE emerge. Although simulation-based testing provides a practicable framework for validation [ASEZ16, BNSB16] and PLE provides the work processes for proper variability management during testing of HCCPSs [CDS07, San16, SKT+16], testing HCCPSs is still time-consuming and optimization methods are paramount. The research work presented in this thesis faces the challenge of cost-effectively testing HCCPSs, by a *feedback-driven, dynamic and iterative test prioritization.*

1.2 Research Methodology

In this thesis we have inspired in the methodology proposed by Vaishnavi [VK04] in order to design and conduct the research activities. The methodology defines an iterative process that consists of five steps. Each process step starts with an input, generates an output product and activates a next step. A general outlook of the research process is summarized in Figure 1.1.

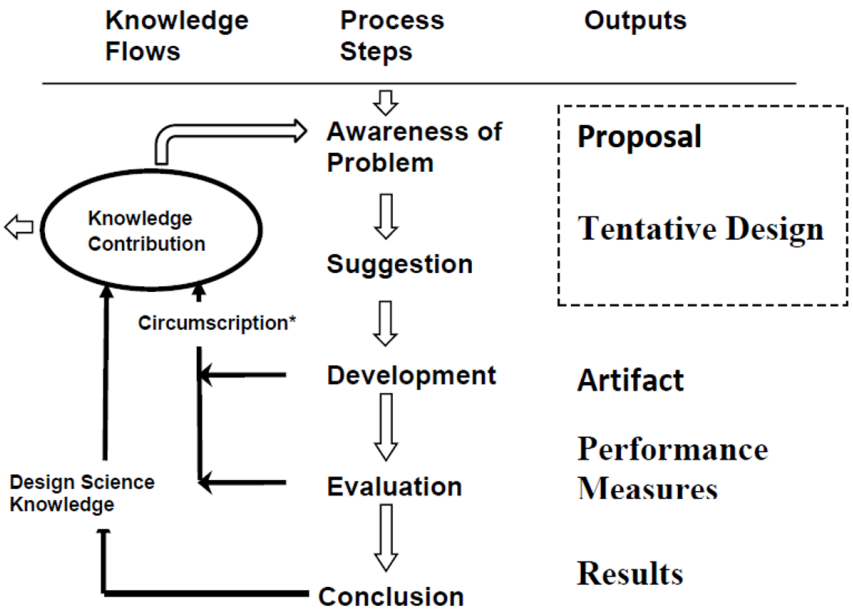


Figure 1.1: The Design Science Research Process Model five steps (Awareness of Problem, Suggestion, Development, Evaluation and Conclusion) and generated outputs summarized in a picture [VK04].

Process steps description:

1. **Awareness of Problem:** The first step aims to identify interesting research problems for current sector specific situations. To achieve this objective, new industry developments or related discipline readings are undertaken. Consequently, a proposal for new research work is formulated as the output.

2. **Suggestion:** The second step is a creative step where new functionality is conceived based on a novel configuration of pre-existing or a combination of pre-existing and new elements. A Tentative Design is obtained as the output.

3. **Development:** This step further develops and implements the Tentative Design producing an artifact as the output. The novelty is found in the design, not in the construction of the artifact. Subsequently, the techniques for implementation vary depending on the artifact to be created.

4. **Evaluation:** In the fourth step, according to the criteria defined in the Proposal, the artifact is evaluated. Consequently, deviations from expectations are identified and could be tentatively explained. At this point, hypotheses are also made about the behavior of the artifact. The evaluation phase results and information from preliminary step might be brought back to the Suggestion step to create another iteration. The output for this step is a set of performance measures.

5. **Conclusion:** The fifth and final step is the end of the research cycle or of a specific research effort. Results of the work are described, categorized and conclusions obtained as the output. This results are reported as Knowledge Contribution.

The methodology can be applied in a iterative way. As is described by authors, "*Development, Evaluation, and further Suggestion are frequently iteratively performed in the course of the research effort*". In this thesis, the methodology has allowed us to approach the research work in an exploratory and iterative manner.

The thesis presented in this dissertation is proposed as a continuation of Aitor Arrieta's thesis [Arr17]. Therefore, the knowledge contributed by his thesis has been an input for the awareness of the problem (the first step of the methodology) for this thesis. In the first step, the aforementioned knowledge was taken into account and additional literature references were analyzed in order to identify the state of the art and delimit the problem. In the second step, a central idea was established as the basis for the proposals, based on the opportunity identified in the literature to solve the given problem.[1] Two iterations of development-evaluation-conclusion steps were conducted during a exploratory stage, to validate intermediate proposals. On the basis of the obtained results from the exploratory stage, finally, a new development-evaluation-conclusion iteration was conducted.

[1]Central Idea: This document is later referred to this ideas as *core concept*, details can be found in Section 4.3.1

1.3 Technical Contributions

This section summarizes the main contributions of the thesis. First, we present the major contribution and secondly a set of two contributions obtained from the exploratory stage are presented.

Most research works address the challenge of optimizing HCCPS testing by focusing improvement on product optimization or test case optimization. At the Domain level, most relevant works propose improvements in the product selection and prioritization steps in order to have a representative subset of products to be tested. At Application level, few notable works have generally been proposed for test case selection and prioritization steps.

Essentially, all contributions of this thesis are aligned with one underlying main idea. Instead of focusing the study on optimizing only one of the previously mentioned steps, the novelty of this thesis consists of performing **joint product and test case selection and prioritization, using information from both the Domain and Application levels**. In addition, each of the contributions brings specific novelties in the employed methods or metrics.

First, the major contribution is presented as follows (further details provided in chapter 7):

- **A novel dynamic test prioritization approach of both product and test cases.** The approach establishes a test plan that executes small groups of test cases with products in iterative executions. After every iteration the approach re-orders dynamically the test plan based on previous executions. The main novelties in this approach consists of (1) the "dynamic fashion" of the re-ordering and (2) the fact of performing the re-ordering of both products and test cases at the same time. The approach is general to any kind of product lines and it has been adapted for testing simulation-based HCCPSs where it has shown to be effective. This contribution has been sent to the Software Quality Journal (SQJO).

A secondary set of contributions obtained during the exploratory stage are presented as follows (further details are provided in chapters 5 and 6 respectively):

- **Search-based fault detection allocating small groups of test cases to products iteratively.** The approach performs product selection and prioritization in a traditional way, to latter apply a novel search-based algorithm that selects a small number of test cases for each of the selected and prioritized products in an iterative manner. The novelty of this approach consists in testing multiple products with a small number of test cases, rather than the traditional exhaustive product testing that performs the testing of each product until all related test cases are employed. This contribution has resulted in the publication of a paper in the International Systems and Software Product Line Conference (SPLC) [MASE17].

- **The comparison of three test case selection methods, based on structural coverage of both the products and the product-line**. The studied test case selection methods consider the employed metrics information of both the Domain and Application levels of the HCCPS. The fact of employing information of both levels is considered the main novelty in this contribution and has resulted in the publication of a paper in the Symposium on Applied Computing (SAC) [MAES19a].

1.4 Publications

This section provides a collection of the works published during this thesis. The publications are scored with the corresponding rankings.

1.4.1 Journal Articles

A journal article was submitted at the Software Quality Journal. By the time this dissertation was submitted this journal paper was in the second round of review.

- U. Markiegi, A. Arrieta, L. Etxeberria and G. Sagardui. "Dynamic test prioritization of product lines: An application on configurable simulation models" in Software Quality Journal JCR: 1.460. Q3. Second round of review.

In addition, during this thesis contribution to other journals papers has been carried out.

- A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, L. Etxeberria. "Employing Multi-Objective Search to Enhance Reactive Test Case Generation and Prioritization for Testing Industrial Cyber-Physical Systems" in IEEE Transactions on Industrial Informatics JCR: 6.764. Q1. DOI:10.1109/TII.2017.2788019
- A. Arrieta, S. Wang, U. Markiegi, A. Arruabarrena, L. Etxeberria, G. Sagardui. "Pareto efficient multi-objective black-box test case selection for simulation-based testing" in Information and Software Technology JCR: 2.694. Q1. DOI:10.1016/j.infsof.2019.06.009

1.4.2 International Conferences

Research work achievements have been published in international conferences. The classification defined by the Spanish Scientific Society of Computer Science has been adopted to determine the ranking of each conference publication.

- U. Markiegi, A. Arrieta, G. Sagardui and L. Etxeberria. "Search-based product line fault detection allocating test cases iteratively". In Proceedings of the 21st International Systems and Software Product Line Conference-Volume A (pp. 123-132). Ranking_SCIE: A-. DOI:10.1145/3106195.3106210

- U. Markiegi, A. Arrieta, L. Etxeberria and G. Sagardui. "Test case selection using structural coverage in software product lines for time-budget constrained scenarios". In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (pp. 2362-2371). Ranking_SCIE: A-. DOI:10.1145/3297280.3297512

Additionally, works indirectly related to the thesis have been published in international conferences and symposiums.

- A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, and L. Etxeberria. "Search-Based Test Case Generation for Cyber-Physical Systems" in CEC2017: IEEE Congress on Evolutionary Computation, 2017, pp. 688-697; Ranking_SCIE: A-. DOI:10.1109/CEC.2017.7969377

- O. Berreteaga, G. Sagardui, L. Etxeberria, U. Markiegi and X. Perez. "Delta Rhapsody" In 26th International Council on Systems Engineering (INCOSE'2016). DOI:10.1002/j.2334-5837.2016.00143.x

- L. Etxeberria, F. Larrinaga, U. Markiegi, A. Arrieta and G. Sagardui. "Enabling Co-Simulation of Smart Energy Control Systems for Buildings and Districts" in ETFA2017: IEEE 22nd Conference on Emerging Technologies and Factory Automation, 2017, Ranking_SCIE: B. DOI:/10.1109/ETFA.2017.8247746

1.4.3 Workshops, Symposiums and National Conferences

In addition to international conferences, two national conference papers were published. Furthermore, two workshop papers were published at ECMSM'17 and SPLC'19. Finally, a Doctoral Symposium paper was published at SPLC'17.

- U. Markiegi "Test optimisation for highly-configurable cyber-physical systems". Doctoral Symposium In Proceedings of the 21st International Systems and Software Product Line Conference-Volume B. 2017. DOI:10.1145/3109729.3109745

- A. Arrieta, U. Markiegi, and L. Etxeberria. "Towards Mutation Testing of Configurable Simulink Models: a Product Line Engineering Perspective" in JISBD2017: XXII Jornadas de Ingeniería del Software y Bases de Datos, 2017. Handle:11705/JISBD/2017/008

- X. Perez, O. Berreteaga, L. Etxeberria, A. Arrieta, and U. Markiegi. "Modeling Systems Variability with Delta Rhapsody" in JISBD2017: XXII Jornadas de Ingeniería del Software y Bases de Datos, 2017. Handle:11705/JISBD/2017/006

- G. Sagardui, J. Agirre, U. Markiegi, A. Arrieta, C.F. Nicolás, and J.M. Martín. "Multiplex: A Co-Simulation Architecture for Elevators Validation" in ECMSM 2017: IEEE International Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics, 2017, 1-6. DOI:10.1109/ECMSM.2017.7945883

- U. Markiegi, A. Arrieta, L. Etxeberria and G. Sagardui. "White-Box and Black-Box Test Quality Metrics for Configurable Simulation Models". REVE 2019: Seventh International Workshop on Reverse Variability Engineering In Proceedings of the 23rd International Systems and Software Product Line Conference-Volume B (pp. 211-214). DOI:10.1145/3307630.3342396

1.5 Related Activities

In addition to attending conferences in the publications of the aforementioned conferences, the PhD student has carried out other activities that have helped him in his training as a researcher. Particularly remarkable among the activities is the participation in European projects.

- "TESTOMAT: The next level of test automation" is a project of the 3rd call ITEA 3 that has been developed between 2017 and 2020. The PhD student has contributed in three technical work packages (WP3, WP4 and WP5) in order to improve test effectiveness through traceability and log models, simulation-based test case prioritization and test automation. Furthermore, the PhD student has carried out the Country Coordinator role at the Spanish consortium, actively participating at General Assemblies and yearly Reviews of the project.

- "SCRATCh: SeCuRe and Agile Connected Things" is a project of the 4rd call ITEA 3 that is been developed since 2018. The PhD student is collaborating with ULMA Embedded Solutions in this project in order to support the automation of secure deployment of highly distributed IoT systems on a continuous basis.

- "VINDICATOR: Accelerating Automated Cloud–Based Testing through Simulation, Modeling and Virtualization" is a project of the ECSEL-2020-2-RIA call. The Ph.D. student has been involved in the proposal writing. Although the commission did not approve funding for the project, the consortium of which we are an active part, it is working on the basis of the feedback provided by ECSEL, to resubmit it in future calls for proposals.

1.6 Document structure

The document is structured into three parts. The first part presents the foundation and context of the thesis with Chapters 1, 2, 3 and 4. The second part describes the work and results obtained from main contribution, which is described in Chapter 7, as well as the work and results obtained from the exploratory research with Chapters 5 and 6. And finally, the third part details conclusions and final remarks in Chapter 8.

1. **Chapter 1 Introduction** describes the motivation for the thesis, applied research methodology and obtained main contributions and publications.

2. **Chapter 2 Technical Background** introduces general technical concepts related to the thesis, such as, CPS, Highly-Configurable System (HCS), SPL, HCCPS and their testing. Additionally complementary notions of Search-Based Software Engineering (SBSE), Mutation Testing and Quality Metrics for testing HCCPSs are introduced.

3. **Chapter 3 State of the Art** presents the state of the art divided into six main sections: (1) SPL testing strategies (2) product selection, (3) product prioritization, (4) test case selection, (5) test case prioritization and (6) critical analysis.

4. **Chapter 4 Theoretical framework** contains the definition of the research objectives and hypothesis as well as the overview of the theoretical framework of this thesis. Finally, employed case studies are introduced and the conducted case study building semi-automation procedure detailed.

5. **Chapter 5 Search-Based Test Allocation for Iterative testing of HCCPS** presents the approach and results of the method developed for allocating small number of test cases in each product for iteratively testing the product line.

6. **Chapter 6 Test Case Selection of HCCPS using Structural Coverage** presents the approach and results of the method developed for selecting test cases based on structural coverage of both Domain and Application levels of the product line.

7. **Chapter 7 Dynamic prioritization of Products and Test Cases for testing HCCPS** presents the approach and results of the method developed for dynamically prioritizing test leveraging information of tests being executed in specific products.

8. **Chapter 8 Conclusion** summarizes contributions, concludes about proposed hypothesis and presents final remarks about work limitations and future works.

Chapter 2

Technical Background

Contents

This chapter presents the basic concepts for this thesis in four sections: (1) Cyber-Physical Systems and its testing, (2) Highly-Configurable Systems, Software Product Lines and its testing, (3) Highly-Configurable Cyber-Physical Systems, its testing and (4) a set of complementary concepts to be taken into account to better grasp the thesis.

2.1 Cyber-Physical Systems (CPS)

This section presents the basic CPS definition, characteristics and application domains. In addition, the section introduces to testing CPS fundamentals.

2.1.1 Basic CPS concepts

The term CPS was coined by Helen Gill at National Science Foundation (2006) to refer to the integration of computation with physical processes. According to the description provided by Lee and Seshia [LS17], in CPSs, embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa.

Definition 1: Cyber-Physical Systems (CPS)

A Cyber-Physical System (CPS) is an integration of computation with physical processes whose behavior is defined by both computational and physical parts of the system.

(Lee and Seshia [LS17])

Following Lee and Seshia's definition CPSs are composed of three main parts as it is showed in Figure 2.1. The *physical part* (denoted as physical plant in Figure 2.1) is the system to be controlled and monitored, for instance, an aircraft, an energy grid, a wheel of a car, etc. The physical part usually includes mechanical parts, biological or chemical processes, or human operators. In the physical part, simultaneous physical processes occur in continuous time according to laws of physics [LBB15]. The *computational platform* consists of sensors, actuators, one or more computers and embedded software. Computational platform controls the physical plant (through actuators in Figure 2.1) and also performs the measuring of the plant (through data provided from sensors in Figure 2.1). Therefore, the behavior of the physical plant is conditioned by the

decisions obtained from the computational platform. And the behavior of the physical plant determines the decisions that the computational plant(s) must make. The *network fabric* provides the mechanisms for the computational platforms to communicate. Lee and Seshia [LS17] describe CPSs not only as a mere union of the two layers but also as an intersection of both since both layers mutually determine each other's behavior.

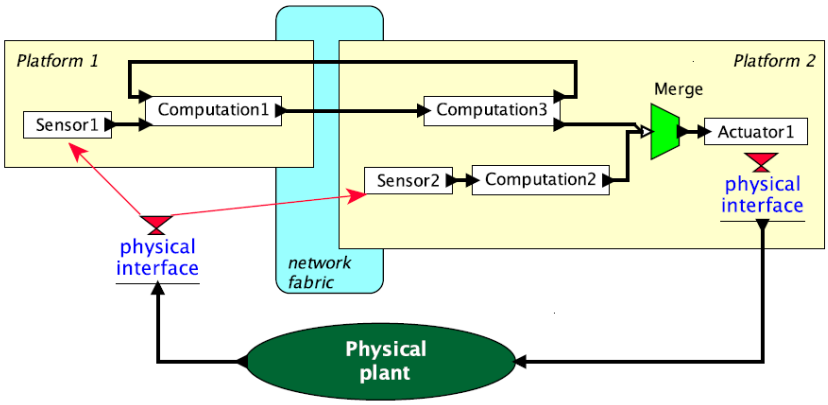


Figure 2.1: Schematic of a simple CPS proposed by Lee and Seshia [LS17]

CPS characteristics

[SWYS11] describe the following **characteristics** for CPSs:

- *Closely integrated:* CPSs are the integration of computation and physical processes.
- *Cyber capability in every physical component and resource-constrained:* Software is embedded in every cyber or physical component. Resources such as computing, network, bandwidth, etc. are limited.
- *Networked at multiple and extreme scales:* CPS include multiple network technologies and are distributed, using a wide variety of device categories and system scales.
- *Complex at multiple temporal and spatial scales:* CPSs are constrained by spatiality and real time.
- *Dynamically reorganizing/reconfiguring:* CPSs have adaptive capabilities.
- *High degrees of automation, control loops must close:* CPSs enhance man-machine interaction providing advance feedback control technologies.
- *Operation must be dependable, certified in some cases:* Reliability and security are often mandatory characteristics for CPSs.

CPS application domains

Lee and Seshia [LS17] enumerate multiples domains where CPSs are widely used, such as, medical devices, military systems, aeronautics, automotive, railway, manufacturing, traffic control systems, power generation and distribution systems, water management systems, robotics and quite a lot of others. All these domains share a technological scenario where physical world is deeply connected to the information world. In this scenario, the CPS term is closely related to other popular terms such as, Internet of Things (IoT), Industrial Internet, Machine-to-Machine (M2M), Cloud Computing and the fog.

According to Rajkumar [RLSS10, RDNK16] the adoption of CPSs systems is being fostered by the convergence of vendors' needs with technological evolution. On the one hand, CPSs vendors have identified a technological gap to build safety critical CPSs correctly, affordably, flexibly and on schedule. On the other hand, the technological evolution is enabling a new generation of smaller and efficient sensors, reduced computing devices, ubiquitous wireless communications and wider internet bandwidth, alternative energy sources and energy harvesting. The convergence of the gap with evolving technologies will enable a change in how people interact with the physical world (through CPSs), just as the internet changed how people interact with one another.

CPS example

The following academic example provides a simplified demonstration of a CPS. The example, titled *Smart Manufacturing Robotics Cyber Physical System* was completely implemented with MATLAB/Simulink© by Mosterman and Zander [MZ16b] following model-based design techniques and is accessible at the File Exchange platform of Mathworks [1].

The example provides a solution for the distributed Towers of Hanoi problem. Specifically, the objective is to move a stack of three colored blocks from one location to the other such that the final stack is ordered according to color and size. A representation of the CPS model is depicted in Figure 2.2.

The *physical part* includes the pick and place machine to provide services of moving blocks through a pneumatic airflow. The *computational platform* is composed of sensors, actuators, processors and embedded software. A set of camera sensors provide the stereoscopic vision to distinguish colors and to detect position of blocks. The slider and nozzle motor actuators enable

[1] https://es.mathworks.com/matlabcentral/fileexchange/38515-smart-manufacturing-robotics-cyber-physical-system

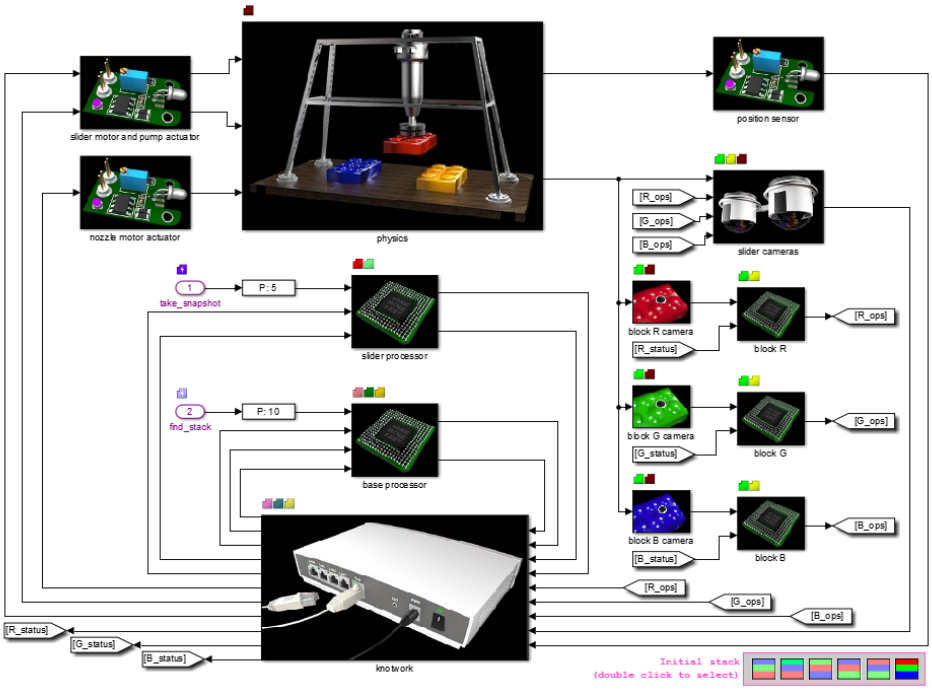


Figure 2.2: *Smart Manufacturing Robotics Cyber Physical System*, a CPS example proposed by Mosterman and Zander [MZ16b])

horizontal and vertical motion respectively. A set of micro processors is provided with the corresponding embedded software to compute color detection and determine required motion. The *network fabric* is composed of a wireless network for communications. Feedback control loop with a time period of 5 ms was established. Refer to original publication for further details [MZ16b].

2.1.2 Testing CPSs

This sub-section introduces the fundamental concepts of CPS testing. It includes (i) relevant methods for testing CPSs, (ii) the different levels of CPS testing depending on the execution platforms and (iii) an introduction to simulation-based testing.

Methods for testing CPSs

The heterogeneous and complex nature of CPSs makes testing such systems a challenging and resource-consuming activity. To address this challenge, paradigms and methodologies from multiple disciplines are employed, as reported in different studies [ZGHY18, AIH15, KDJ+16, ZN08]. For instance,

Zhou et al. [ZGHY18] reported a classification of CPS testing methods including model-based testing, search-based testing or fault injection-based testing among others. Asadollah et al. [AIH15] classified CPSs testing methods according to the V-model levels for validation and verification (i.e., hardware-software-network testing at unit level, integration testing, system testing, etc.). A multi dimensional classification was proposed by Zander [ZN08], where testing activities were sorted according to defined five dimensions: test goal, test scope, test reactiveness, test abstraction and test execution platform.

Test Execution Platforms for CPS testing

Test Execution platforms are the systems in charge of managing the execution of tests. For this purpose, testing platforms have a number of inputs and outputs for the System Under Test (SUT) (i.e., system to be tested). By stimulating the inputs, the behavior of the SUT is analyzed observing the outputs.

Model-based verification and validation of Embedded Systems rely on *Test Execution Platforms* to enable a multi-stage testing of complex systems [SH09]. To this end, different scenarios are implemented to gradually integrate the physical aspects of the system.

Shokry and Hinchey [SH09] defined four testing levels, supported by specific Test Execution Platforms. The four testing levels, depicted in Figure 2.3, are denoted as: Model-in-the-loop (MIL),Software-in-the-loop (SIL),Processor-in-the-loop (PIL)and Hardware-in-the-loop (HIL). MIL is performed to analyze the controller model along with the simulated model and to generate the reference test results values for subsequent tests. In SIL, the model is replaced with the executable object code of the generated software. Following this, in PIL the generated software is cross-compiled and executed in the target processor for detecting errors, such as, code generation bugs or compiler related bugs. And finally in HIL, the generated software is deployed in the real-time infrastructure to validate performance requirements and time constraints. For further details of the X-in-the-loop testing levels refer to the original publication [SH09].

Model-based and test execution platforms have shown to be effective to reduce the effort of different verification and validation stages [ZN08]. However, these embedded system testing proposals must be adapted to the CPSs context [MZ16a] where intrinsic parallel task are executed (e.g., concurrent computing tasks and physical processes). The parallel task execution, requires from the

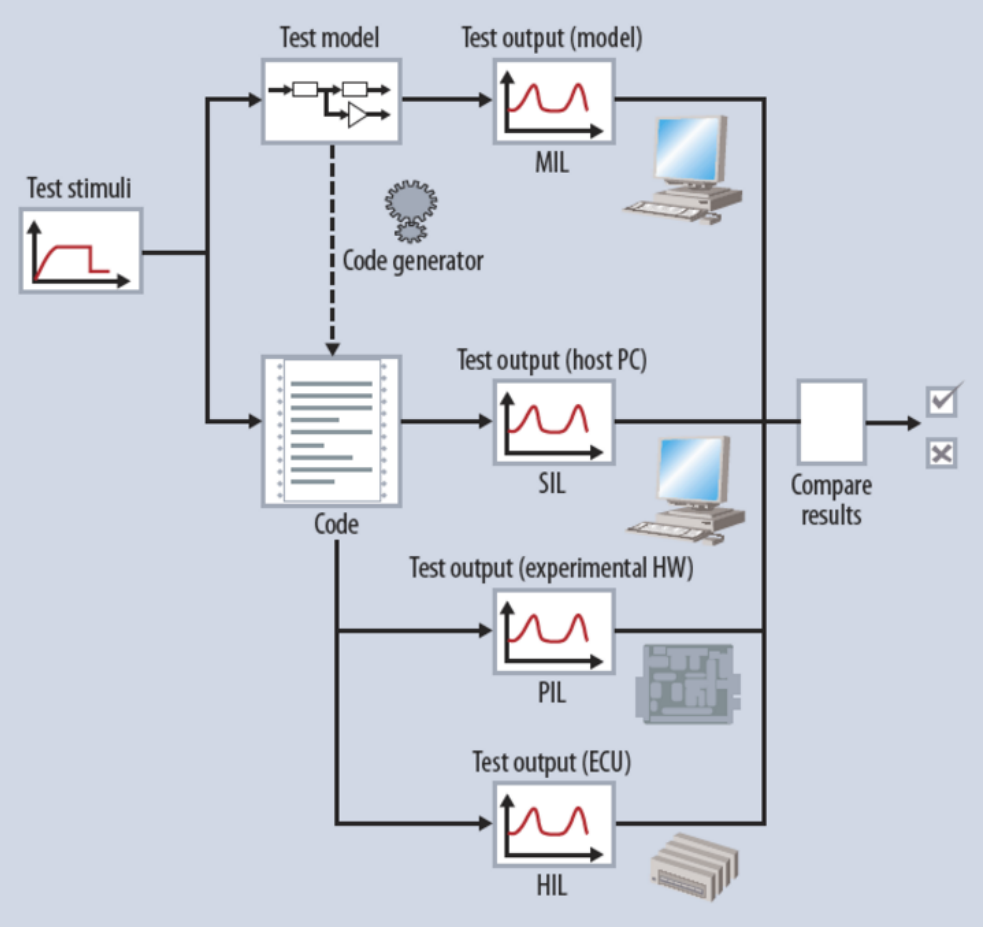


Figure 2.3: X-in-the-loop testing phases, where test reference results are generated in the MIL test and then compared with the corresponding output from all subsequent tests [SH09].

computational platforms a simultaneous manage [LS17], and similarly, testing CPSs must monitor all parallel tasks. Additionally, testing must manage the reconfigurable capabilities of the CPS which handles the unpredictable physics behavior [SWYS11]. In this context, the research and industry communities have proposed model-based methodologies for the particular field of CPSs [JCL11, NAY17] which are mainly relying on simulation-based for the verification and validation stages.

Simulation-based CPS testing

Simulation models provide a virtual representation of a real-world system to experiment with the product at early stages [MNBB16]. Engineers addressing the development of new complex systems, such as CPSs, typically combine

simulation tools with model-based design methodologies [PMB+12, CMM+18]. Resulting configurable simulation models consist of blocks, structured in nested subsystems. Each of the blocks acts according to the inputs to perform the operations for which it is designed and generates the corresponding outputs. For instance, the model depicted in Figure 2.4 represents a cruise controller of a car, composed of six input ports (enable, brake, set, speed, inc and dec) and two output ports (throt and target) structured in two hierarchical levels.

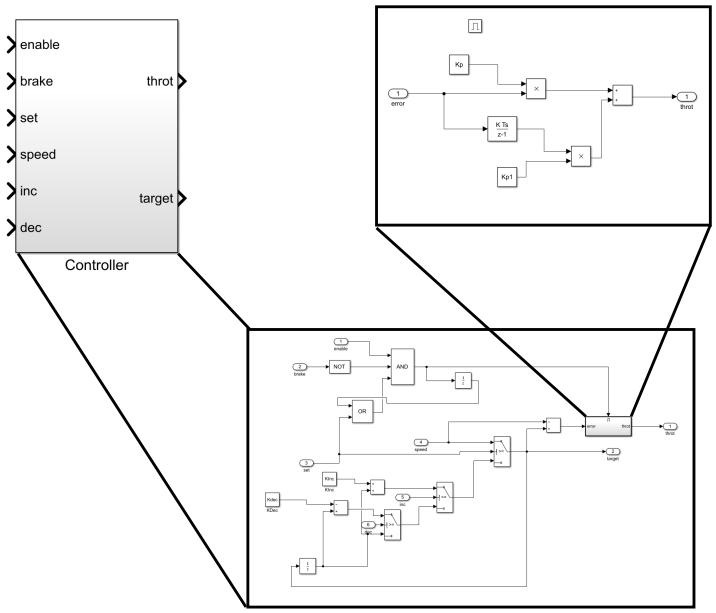


Figure 2.4: Example of a simulation model of a Cruise Controller of a car. The simulation model is composed of six input ports and two output ports and is structured in two hierarchical levels.

Testing CPSs with simulation models has been successfully adopted by both industry and academia, particularly for identifying faults at early stages [MNBB19]. Several tools are available for simulation [UL10], for instance MATLAB/Simulink [Mat21] provides an interesting tool set for CPSs engineers as enables capturing dynamic behavioral models (Refer to Section 4.4.1 for further details of MATLAB/Simulink).

Simulation-based testing typically follows multiple stages at different X-in-the-loop testing levels [SH09]. According to field engineers' experience [AWSE19], different tests are conducted at each level. Moreover, tests are executed multiple times at each level incrementally, starting with low fidelity models to conclude with the most precise ones.

Simulation-based testing of CPS has become decisive in domains where software and physical processes interact [MNBB16]. Two main benefits are obtained from simulation-based testing: firstly, it significantly reduces the cost of creating real prototypes. And secondly, simulation-based testing speeds up early validation tasks. Briand et al. [BNSB16] concluded that *simulation models* which represent aspects such as system behavior, environment, structures and properties of CPSs are capable of raising the level of abstraction at which testing is performed. Employing simulation models permits software engineers to (i) execute more test cases, (ii) develop test methods to select scenarios that should also be executed on the deployed system based on the risk level and (iii) specify test oracles for the automatic fault detection. Moreover, simulation permits testing scenarios that could be dangerous, expensive or even impossible to reproduce employing a real prototype. The aforementioned benefits of using simulation models has fostered its adoption in multiple domains (e.g.,Automotive [MNBB16, ANBS18, AWSE19], Transport [SEA+17], Aerospace [MNBP20], Healthcare [SF12]).

Since testing CPSs with physical prototypes and obtaining an adequate product quality may be unfeasible, a *simulation-based testing* scenario has been employed in this thesis, allowing different testing configurations within the MIL test level execution platform, as previous studies have reported to be effective [ZN08].

2.2 Highly-Configurable Systems (HCS)

This sections describes HCS concepts, relevant HCS applications and HCS development strategies. Additionally, this section presents the basic Software Product Line (SPL) concepts and definitions and introduces to testing SPL fundamentals.

2.2.1 Basic HCS concepts

History of physical goods production has changed considerably over time, as described by Apel et al. [ABKS16]: from handcrafted production in the pre-industrial era, through the mass production of the industrial era to the (current) era of mass customization. This evolution, that has occurred in about a century and a half, has also been reproduced in the software and systems industry, but only in six decades. Early software products (in the 60s and 70s) were handcrafted for specific hardware and sold in small quantities. Later (in the 80s and 90s) the software industry agreed on the development of standard platforms, which allowed the commercialization at scale of software packages that guaranteed common functionalities and quality levels to all customers. However, software standardization was not able to met the particular needs of some specific market niches (e.g., small customers or resource-constrained environments -such as embedded systems-). In response to the unmet needs and the increased demand for customization (aligned with the mass customization of goods production), since 90s proposals to develop reusable systems that could be customized gained importance.

Definition 2: Highly-Configurable System (HCS)

HCS are those systems that allow functional and non-functional properties to be tailored to the requirements of each customer by means of configuration options.

HCSs are composed of common characteristics (i.e., characteristics shared by all systems) and a set of optional characteristics (e.g., alternative hardware components, software functionalities, etc.). Thus, configurable systems offer the possibility for different types of users (e.g., developers, operators or end users) to select optional characteristics when configuring systems to meet their needs.

HCS applications

HCSs are present in many software applications, such as database management systems [BBG+88, RALS09], operating systems[BSL+10, SLB+11], web and content management servers [SSPRC15, SGAK15] and many other system tools and utilities [KSK+19, CGHX19, SGS+15, HZS+16, FMR+20]. HCSs are also present in applications related to embedded systems [SKN+11], mobile phones [OMR10, ZSM17], cyber-physical systems [KNK+17, LYA20, RZ21], robotics [GSB+19] or cloud computing [KKW+18, AKM20]. Relevant experiences of HCSs can likewise be found in the industry [SKN+11, HZS+16, SKT+16, BSZ+20] applied to a wide range of domains (e.g., automotive, aerospace, industrial automation, energy, transportation, etc.).

HCS example

A well-known example of large and complex highly-configurable software is the kernel of Linux operating system family. It provides more than 14,000 build-time configuration options to support multiple functionalities. It can be configured to perform a large set of tasks on different devices (as it is supported in 26 different hardware architectures), ranging from embedded systems, through desktop to servers and clusters in huge data centers. Further details of the Linux kernel can be found in existing large literature [BSL+10, PQM+18, ESYES19, MWVK20].

HCS development strategies

Undertake design, development and operation of HCSs to support multiple variants has been addressed by researchers and practitioners adopting different strategies [BSZ+20]. The *Clown & Own* strategy consists of copying an existing system and adapting it to new requirements [DRB+13]. Formally, it does not perform any kind of systematic variability management. Although it is very widespread due to its rapid initial implementation, it presents scalability and maintenance problems. In the *parameter-based configuration*, system is configured by selecting values for each of the required parameters. For instance, a configuration can be obtained by conditional `#ifdef` directives in source code, which are solved by pre-processor during build process. This approach is common in the industry in general and in embedded systems field in particular [SKN+11, RALS09]. Some fields have developed their own configuration languages, such as the `Kconfig` or *Component Definition Language*

for the Linux kernel [BSL+13]. Nevertheless, *Product Line Engineering (PLE)* deserves to be highlighted as the strategy that allows to effectively design, develop and operate HCSs. This strategy takes advantage of the reusability achieved when implementing a set of systems as one integrated HCS [CDS08]. It has been widely studied by both the academic community and practitioners [RSB+18, CLGGB20]. Next Section 2.2.2 introduces basic SPL concepts).

2.2.2 Basic SPL concepts

Over the last few decades, the evolution of technology and the growing needs of the market to cope with mass customization have driven new opportunities for software development. According to Pohl et al. [PBL05] the main challenges for these opportunities are the reduction of development costs, the enhancement of quality and the reduction of time to market. To deal with the aforementioned challenges, engineers developed the SPL paradigm. SPL is one of the most relevant paradigms of software reusability, providing engineers with the possibility to rapidly configure customized products instead of repeatedly developing new products from scratch.

SPL Definition

Essentially, a SPL is a family of related software systems that shares a common set of features, such as the abstract representation of reusable software increments [BSR03]. SPL conceptually combines off-the-shelf software and custom software. SPL software is designed for a domain (i.e., similar to off-the-shelf software), and to address the needs of various specific market problems (i.e., similar to custom software) by using variability concepts to generate particular software products.

At the beginning of the century, definitions of SPL were provided:

Definition 3: Software Product Line

A SPL consists of a product line architecture, a set of reusable components and a set of products derived from the shared assets.

(Bosch [Bos01])

Definition 4: Software Product Line

A SPL is a set of software-intensive systems sharing a common, managed set of features which satisfy the specific needs of a particular market segment or mission and which are developed from a common set of core assets in a prescribed way.

(Clements and Northrop [CN01])

SPL Engineering

SPL implies a new engineering approach called Software Product Line Engineering (SPLE). Unlike the traditional approach of building products one by one, in SPLE products are assembled by selecting existing assets from the SPL. In recent decades, the field of SPLs has been intensively studied [HPMFA+16] and the SPLE approach has been adopted in multiple professional domains such as automotive, aeronautics or telecommunications to increase quality and reduce development costs and time to market [LSR07, ABKS16, BSL+13, ALHR17].

Pohl et al. [PBL05] proposed the SPLE framework that divides the SPL development into two processes; (i) the Domain Engieering (DE) process and (ii) the Applictaion Engieering (AE) process. The framework (depicted in Figure 2.5) has been widely adopted by the research and industry communities.

Definition 5: Software Product Line Engineering

Software product line engineering is a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customization.

(Pohl et al. [PBL05])

The DE process allows the development of both the common and the variable parts of all products. It is divided into five sub-processes: (i) definition of the product line scope, (ii) eliciting and documentation of common and variable requirements, (iii) design reference architecture, (iv) implement reusable software components and finally, (v) test reusable software components. During the executions of the DE sub-process six main artifacts are generated: (i) the product roadmap artifact that describes the major features of all applications, (ii) the variability model artifact that describes the variability of the product line, (iii) the domain requirements artifact which defines the

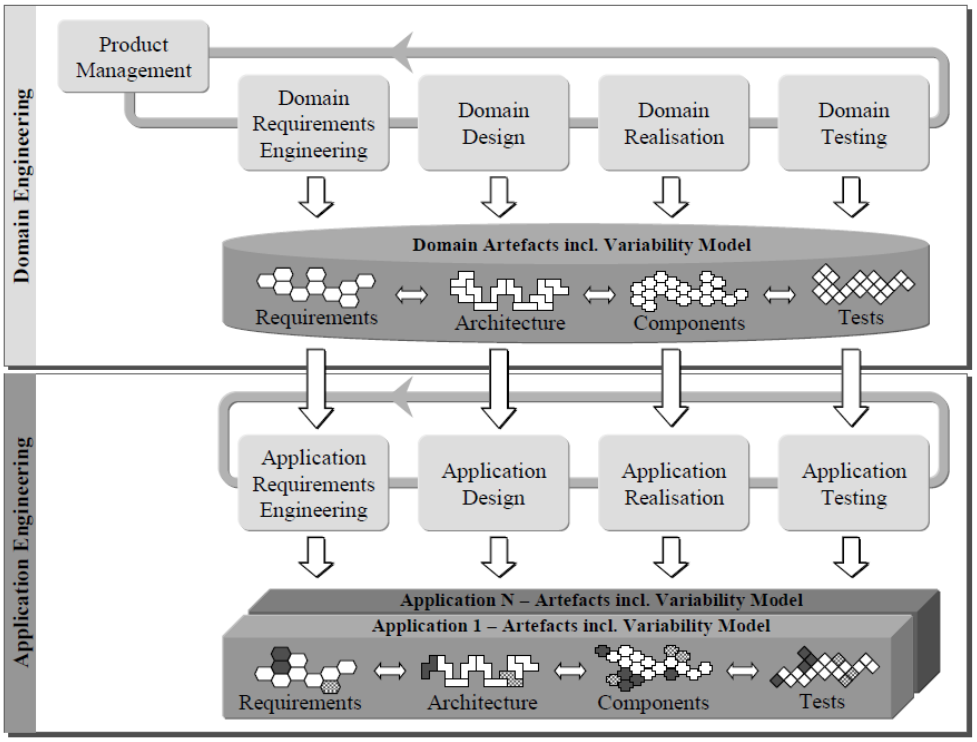


Figure 2.5: SPLE framework proposed by Pohl et al. [PBL05])

reusable requirements, (iv) the domain architecture artifact that defines the reference architecture of core and complementary structure, (v) the domain realization artifact which details the design and implementation, and finally, (vi) the domain test artifact, where reusable tests are designed and planned.

The AE process allows the development of each particular product specifying the use of variable parts. It is divided into four sub-processes: (i) specification of particular product requirements identifying differences from the domain requirements, (ii) definition of how the product will be specialized from the reference architecture, (iii) realization of component configuration and development, and finally, (iv) product validation and verification against its specification. The executions of the AE process generates five main artifacts: (i) the application variability model artifact provides variability binding for applications, (ii) application requirements artifact details the complete specification, (iii) application architecture artifact that specifies the instance of reference architecture, (iv) application realization artifact details the configuration parameters for application-specific realization, and finally, (v) application test artifact which provides the complete test documentation.

Variability

In the context of SPLE, **variability** is defined by Pohl et al. [PBL05] as the possibility to determine particular artifacts of DE that not necessarily are part of each product at AE. For that purpose, **variation points** are established specifying the type and location of the variability. Moreover, possible assignments for each variation point are described, at least one per variation point. Each assignment is named **variant** and can contain within nested variation points.

Definition 6: Variability

The possibility to determine particular artifacts of Domain Engineering that not necessarily are part of each product at application engineering.

(Pohl et al. [PBL05])

Variability management is a key activity related to all SPLE framework subprocesses. Thus, variability representation gains importance when engineers must handle variability. Numerous approaches have been proposed to represent variability, including the Orthogonal Variability Model proposed by Pohl et al. [PBL05] and several approaches based on decision model, such as, Synthesis, KobrA, Dopler or VManage reviewed at [SRG11]. Nevertheless, Feature Models are recognized in literature as predominant variability representation for SPL.

Feature Model (FM)

Feature Models (FMs) are employed to represent all possible products of a SPL based on features and relationships among them [BSC10]. FMs are visually represented with FM diagrams as a hierarchical structure set of features and constraints. Features represent an increment in product functionality [Bat05] and constraints define restrictions to the usage of features. When designing FMs, the features are arranged in a tree-like hierarchy representing the parent-child relationships between features. In addition, FM constraints provide compositional rules that cross-tree checks mutual dependencies and exclusions between features [KCH+90]. There are two types of constraints. The *Requires* constraint means that when a feature is selected within a product, it is mandatory to select the required related feature. The *Excludes* constraint means that when a feature is selected, the excluded related feature cannot be selected. Constraints can be represented by dashed lines that identify features

involved in the rule. When a product is configured, a set of features are selected satisfying FM constraints. A simplified FM example of mobile phones is depicted in Figure 2.6.

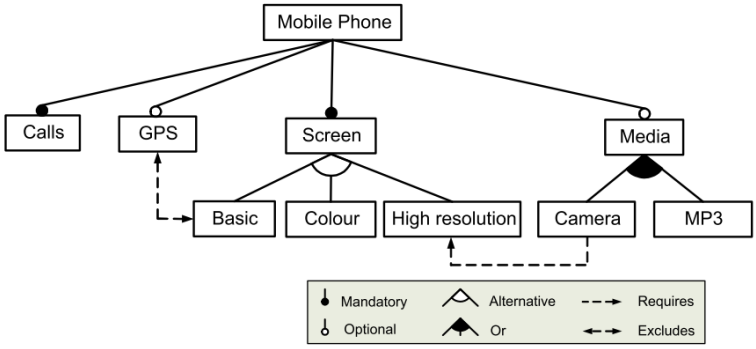


Figure 2.6: Simplified mobile phone FM sample [BSC10]

Consider the example of Figure 2.6 where a simplified feature model from the mobile phone industry is depicted [BSC10]. The features are used to specify different possibilities to configure mobile phones (products). According to the model, all products must include call support (i.e., `Calls` feature), as well as any of the screen options provided (i.e., `Basic`, `Colour` or `High resolution` screen type). Optionally, the product may include support for `GPS` and multimedia devices (i.e., `Camera`, `MP3` player or both of them). When the `Basic` screen feature is selected, the `GPS` feature is excluded due to the constraint among both features. However, if the `Camera` feature is desired, it is required to include a display with the `High resolution` feature to satisfy the existing constraint.

Definition 7: Feature Model

Feature Model represents the information of all possible products of a Software Product Line based on features and relationships among them.

([BSC10])

The Feature Model term was first introduced by Kang [KCH+90] as part of the Feature-Oriented Domain Analysis (FODA) report in 1990. Since then, many different FM representation approaches and extensions have been proposed, being the most relevant: the basic feature models, cardinality based feature models and extended feature models. Further detailed readings are referred to [BSC10, San16].

Definition 8: Configuration

A configuration is a selection of features for product instantiation. Thus, a configuration only allows feature combinations that could lead to a product and do no infringe relations and constraints between features.

On the basis of a given FM, the defined feature selection to instantiate a product is named configuration. The configuration is feasible only if feature relations and constraints are not infringed. When a defined configuration is implemented the ready-to-use particular product is generated.

Definition 9: Product

A product is a selection of features including their implementation resulting in a product ready to be used.

2.2.3 Testing SPLs

Testing single systems is not a trivial activity. Although the objective is simple, to detect as many so-called bugs as possible, it is not usually feasible to test the whole system due to limited resources. The challenge becomes therefore to effectively identify and execute the tests that will allow us to detect as many bugs as possible [AO16, FR19]. Testing SPLs is even more challenging due to the variability [TTK04]. Benefits gained from the potential provided by variability during SPL development become into difficulties and risks of the same magnitude to be managed when facing testing. For instance, an undetected error in a component that is widely reused will be propagated to all products in which it is employed. Main challenges identified when testing SPLs are (i) the large number of tests, (ii) balance between test effort for reusable components and concrete products, and (iii) handling variability in testing [ER11].

Different initiatives have contributed from (single system) software engineering to transfer knowledge and experience to the product-line community so as to establish a solid basis for testing SPLs [McG01]. Moreover, new initiatives focused on the product-line community's particular needs have been introduced [PBL05]. The SPL testing research and practitioners community is very active, what is reflected in the vast number of publications and the regular literature revision. As a result the state of the art and practice in terms of test levels,

test approaches and test strategies are frequently reviewed. Refer to Chapter 3 for further details with regard to SPL testing strategies.

SPLE related testing activities

Considering the framework for SPLE proposed by Pohl et al. [PBL05] (refer to Figure 2.5 for further details), two testing related activities are identified: Domain Testing (DT) and Application Testing (AT). The former is the sub-process of DE where evidence of defects in domain artifacts are uncovered and where reusable test artifacts for application testing are created. The latter is the sub-process of AE where domain test artifacts are reused to uncover evidence of defects in a specific application.

Due to these two testing activities, product line testing also includes an additional challenge: the dimension of what should be tested in DE and what should be tested in separate products at AE [ER11].

SPL testing levels

There are several Software Development Life-Cycles (SDLCs) models to address verification and validation activities. A widespread model (among embedded systems and also in CPSs) is the V-Model (originally named *V-chart* [Boe79]). The V-Model proposes the creation of test artifacts at different levels (i.e., unit test, integration test, system test and acceptance test) associated to the requirements, design and corresponding code [Boe79]. The V-Model proposal for single system requires additional activities in the SPL context, since testing is divided into two sub-processes: Domain Testing (DT) and Application Testing (AT). To this end, the *extended V-Model* was defined [JHQJ08]. The extended V-Model, depicted in Figure 2.7, overlaps two V-models to accommodate both Domain and Application testing activities.

It should be noted, as Pohl et al. [PBL05] points out, that in DE it is not possible to perform testing of all artifact due to the variability it is not feasible generating or adapting artifacts to the particularities of the products. Moreover, although testing in AE is similar to the V-Model of single system, it should be noted that testing artifacts may already be pre-generated (for the common part) and tested, whereas artifacts for the variable part may already be generated but will still require adaptations before they can be used. The detailed SPL testing life-cycle activity is established when defining the SPL testing strategy.

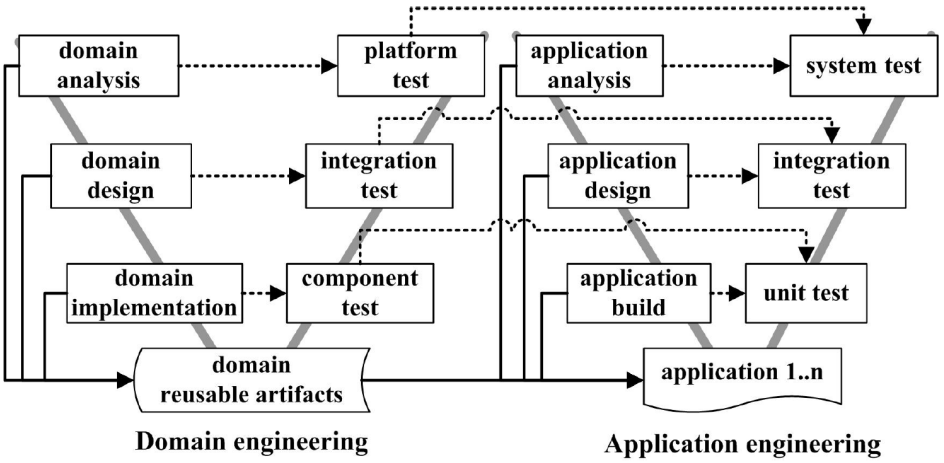


Figure 2.7: SPL testing life-cycle extended V-Model [JHQJ08])

SPL testing strategies

There are multiple factors that may condition the selection of the SPL testing strategy (e.g., number of products to be derived, required learning effort for test engineers, required time to create test artifacts, etc.). Pohl et al. [PBL05] classifies SPL testing strategies into four fundamental types:

- *Brute Force Strategy (BFS):* All activity is performed exclusively in DE. To this end, the strategy performs all test activities at all test levels and for all possible applications during Domain Testing.
- *Pure Application Strategy (PAS):* Testing is performed exclusively in AE. Test artifacts are created specifically for particular applications (i.e., it does not generate reusable test artifacts).
- *Sample Application Strategy (SAS):* One or a few sample products are created to early test DE artifacts. Since not all possible applications are tested in DE, application testing must be also performed in AE, where some DE test artifacts may be reused.
- *Commonality and Reuse Strategy (CRS):* In DE common parts are tested and reusable test artifacts are prepared for variable parts. In AE common parts are tested with domain artifacts (to ensure correctness in the particular application) and variable parts are tested adapting reusable test artifacts.

Other classifications exist that categorize strategies in slightly different ways, with nuances that have been pointed out by the community. For further details refer to Section 3.1 where state-of-the-art of SPL testing strategies is provided.

This thesis addresses the test optimization of configurable CPS systems, assuming that Configurable CPSs are HCSs. When managing the variability of configurable CPSs, we adopt the practices of the product-line engineering community.

2.3 Highly-Configurable Cyber-Physical System (HCCPS)

This section presents the definition of HCCPS. In addition, variability management and specific particularities when testing HCCPSs are highlighted. Finally, the optimization of HCCPS testing is introduced.

2.3.1 Basic HCCPS concepts

The complexity of CPSs is increasing to meet the technological challenges that need to be addressed (e.g., autonomous vehicles are in the spotlight as a relevant research topic [SER21]). Furthermore, there is an growing need to incorporate variability to the system in order to meet the demand for customization of different market segments. Therefore, we call *Highly-Configurable Cyber-Physical System* to those CPS systems that integrate multiple technologies (e.g., mechanical, electrical, software, etc.) and are developed as variant-rich HCS which can be configured in many different ways.

Definition 10: Highly-Configurable Cyber-Physical System (HCCPS)

A Highly-Configurable Cyber-Physical System (HCCPS) is CPS that integrates multiple technologies and can be tailored to the requirements of each customer by means of configuration options.

2.3.2 Testing HCCPS

Testing HCCPSs address the challenges of testing both CPS and HCS worlds. It therefore requires solving both variability management as well as the particularities derived from the multidisciplinary nature of CPSs.

Variability management in HCCPSs

Owing to the complexity introduced by the variability management, Arrieta et al. [AWSE16b] stated that development and validation of these HCCPSs can be similar to the processes defined for the development and validation in PLE. Thus, at DE process, the development of common and variable parts of the CPS products is undertaken. These product parts are later reused at the AE process, where the variability is resolved and specific CPS products are generated.

Differences between testing HCCPSs and SPLs

PLE are very often oriented to software. However, the intrinsic differences between HCCPSs and SPLs must be taken into account when testing. Table 2.1 summarizes main differences between HCCPSs and software product line testing.

Table 2.1: Testing differences between HCCPSs and SPLs.

Highly-Configurable CPS	**Software Product Lines (SPL)**
Cost of the prototype	No prototype required
High test execution time (Sw. and Physical layers simulated)	Lower test execution time
Many domains (often co-simulation required)	Mainly software domain
Faults at software, interaction, sensors, actuators, communication systems, etc.	Faults at software, interaction faults, etc.
MIL, SIL, PIL, HIL	MIL, SIL

As in SPLs, the elevated number of system variants that can be configured in HCCPSs (i.e. thousands or even millions of system variants), makes unfeasible to test all of them. Moreover, an unclear notion of test coverage is obtained. Thus, testing HCCPSs requires optimizing the invested testing time while guaranteeing a high overall test quality. Although both HCCPSs and SPLs systems require high amount of testing time, HCCPSs requires extra time to execute the software and physical layers simulation. Furthermore, in HCCPSs additional time must be considered due to the co-simulation processes time consumption. Another remarkable difference is related to the prototype, while HCCPSs requires prototype which increments the cost of its building, SPLs does not require it. According to covered fault types, SPLs looks for software and interaction faults, whereas HCCPSs must also take into account faults coming from physical layer elements (i.e., sensors, actuators, communication elements, etc.). Finally, while SPLs only requires testing at MIL and SIL levels, HCCPSs must add the PIL and HIL levels.

2.3.3 Optimization of HCCPS testing

The optimization of HCCPS testing requires to address challenges of both CPS and HCS. In the particular case of CPS, in this thesis, we limit CPS testing to the MIL level with implementations built on simulation-based testing (this aspect has been detailed in Section 2.1). In the HCS working plane, we rely on the most common practices when optimizing SPL testing.

Existing works may be grouped into four major categories when addressing SPL test optimization. Firstly, most of the works focus on *product selection*, so that a representative subset of products from the product line can be obtained or sampled. Improvement in this aspect has a significant impact on optimization by substantially reducing the number of products to be tested (i.e., from thousands to tens). Secondly, with *product prioritization*, the optimal ordering of products aligned with testing objectives is established (e.g., to detect faults as early as possible). Thirdly, the optimization of test case selection has been studied to increase the efficiency of testing. Lastly, in a fourth group are works that study optimization based on *test case prioritization*. Thus, optimization of SPL testing can involve optimization in each of the four categories: product selection and prioritization and test case selection and prioritization. Refer to Section 3 for further details.

This thesis focuses on CPS testing at the MIL level using simulation models. In the HCS area, the thesis addresses SPL testing optimization categories, by adopting the Sample Application Strategy.

2.4 Complementary notions

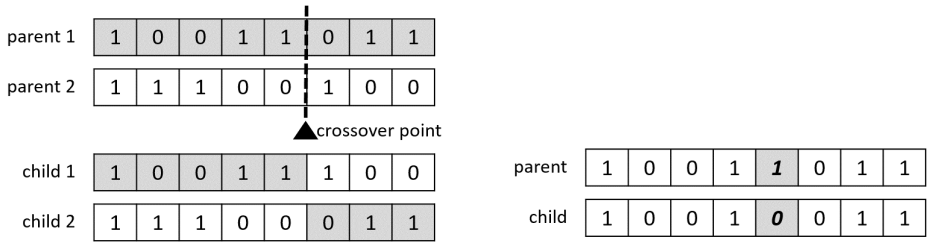
When designing and developing the experimentation of the contributions, different techniques and metrics (not described in this chapter so far) have been used. This section presents basic concepts to better understand the work undertaken. Specifically, the concepts of (i) Search-Based Software Engineering, (ii) Mutation Testing and (iii) Quality Metrics for testing are introduced. First, Search-Based Software Engineering is an optimization technique employed in the first exploratory work. Secondly, Mutation Testing consists of a testing technique based on artificial faults that we employed in our case to measure the effectiveness of the approaches proposed in all the works. Finally, different Quality Metrics for testing are introduced, based on both white-box and black-box, some of which have been employed in this thesis.

2.4.1 Search-Based Software Engineering (SBSE)

Test optimization problems requires the management of large complex problem spaces with multiple competing and conflicting objectives. To handle such challenges, Search-Based Optimization (SBO) has demonstrate to be an effective collection of techniques from the metaheuristic search, operations research and evolutionary computation paradigms in multiple areas [HMZ12]. SBSE is the discipline that focuses on the application of SBO techniques to software engineering problems [HMZ12].

Some of the most popular techniques that can be applied in SBSE are: Hill Climbing, Simulated Annealing, Greedy or Genetic Algorithm (GA). Among them, GA is the one of the most famous Evolutionary Algorithm technique [LHLE15]. To implement these techniques specific algorithms are provided. For example, the following three algorithms are different implementations of the GA technique: (i) Weigh-Based Genetic Algorithm (WBGA), (ii) Random-Weighted Genetic Algorithm (RWGA) and (iii) Non-dominated Sorting Genetic Algorithm II (NSGA-II).

In the specific case of GA, algorithms mimic natural selection process to search optimal solutions for diverse optimization problems [WAG15, WAY+16]. To guide the search, a mathematical objective function is defined named *Fitness Function*. To implement Genetic Algorithms, an initial population is randomly generated. The algorithm then applies three operators: (i) a selection operator which evaluates each individual to identify the best ones, (ii) a crossover operator, which represents mating between parent individuals to generate new



(a) Crossover operation example where parent individuals generate children individuals by exchanging remaining elements after the crossover point.

(b) Mutation operation example where a parent individual is undergone a muting change of one element generating a new individual.

Figure 2.8: Sample crossover and mutation genetic algorithm operations.

child individuals and (iii) a mutation operator, which introduces modifications to parent individuals, generating new child individuals. Examples of crossover and mutation operators are provided in Figure 2.8. The crossover operation example swaps elements of two parents members starting at the crossover point to generate the children in Figure 2.8a. The mutation operation is exemplified in Figure 2.8b by modifying an element of the parent to create the mutated child. In the resulting population after the three operators execution, only individuals that fit the best to the problem are kept. This way a population generation cycle is completed. These cycles are repeated until the predefined number of generations is reached.

Four variants of GA techniques are applied in the exploratory contribution described in Chapter 5 to propose algorithms that optimize test case allocation to products.

2.4.2 Mutation Testing

We distinguish two parts when describing Mutation Testing. First, we introduce the basic concepts of Mutation Testing for general software systems. Secondly, we describe how Mutation Testing is approached in systems that are based on simulation-models.

Mutation Testing basic concepts

According to the definition provided by Jia and Harman [JH10] *"Mutation Testing is a fault-based testing technique"*. The underlying principle of mutation

testing is to emulate the mistakes that programmers usually make to measure the effectiveness of our tests to reveal those artificial faults. To this end, faults (a.k.a. *mutation*) are deliberately seeded into a copy of the original software by making slight syntactic changes. The faulty copy of the software is called *mutant*. When we stimulate both the original and the mutant copies of the software with the inputs of a test case, if test results are different it is considered that the fault has been detected (a.k.a. *'killed'*).

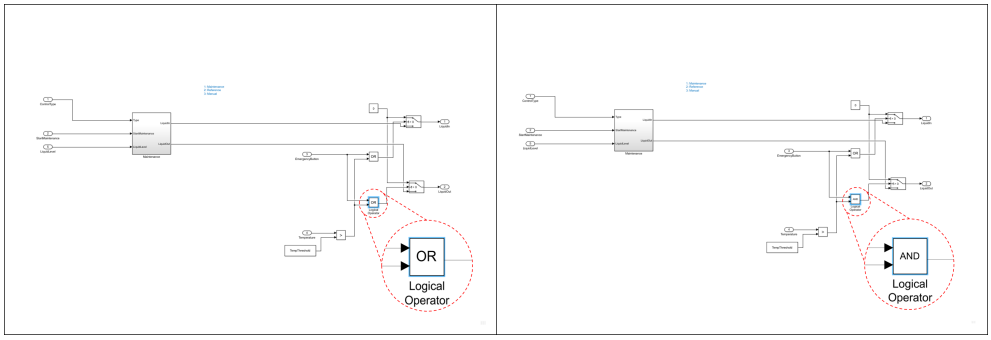
In the same way that exists different types of faults, different types of *mutation operators* have also been defined. For instance, let us consider the following statement `if (x>0 || y>0) result = true;`. The *logical replacement operator* injects a mutant when we replace the || (OR) logical operator with the && (AND) logical operator, obtaining a faulty copy with the following statement `if (x>0 && y>0) result = true;`. However, the challenge is not using abundant mutation operators, but their efficient selection, known as *mutant reduction*. A comprehensive classification of both mutation operators and mutant reduction strategies is provided by Papadakis [PKZ+19].

The mutation testing technique provides a testing criterion named *Mutation Score*, that measures the ratio between the number of detected faults over the total number of the seeded faults. The more of detected faults, the greater the mutation score is.

Mutation testing can be used to test software at any level (i.e., unit, integration, system, etc.) and has been adapted to a large number of technologies and programming languages [JH10]. Both academia and industry have largely researched the technique during last decades [JH10, PKZ+19] and is considered a mature technique as it is increasingly adopted in multiple domains [PKZ+19].

Mutation Testing in simulation models

Basic concepts described for software systems are transferable to the working environment of simulation models. Therefore, seeding artificial faults to assess the quality of test suite is also valid in this field. Mutation operators employed for software systems can also be used in the software layer of the CPS modeled through a simulation model. Let us review the example in Figure 2.9. On the one hand, the model on the left (Figure 2.9a) shows the control subsystem (cyber part of the CPS) of the Industrial Tanks case study (for more details refer to Section 4.4). Note that the block labeled *LogicalOperator* (magnified in the red circle) is set to the *OR* value. On the other hand, the model on the right (Figure 2.9b) is a copy of the original system, where a Logical Operator



(a) Original model **(b)** Mutated model

Figure 2.9: A mutation operation example on model.

Replacement (LOR) fault has been seeded. In this case, the *LogicalOperator* block is set to *AND* value. Apart from software mutation operators, new types of mutation operators emerge to test also the hardware layer or specific aspects of the simulation model. For instance, a mutant on hardware can be seeded replacing the Variable Change Operator (VCO) of a Gain type of block, or a Delete Transition Operator (DTO) can be seeded in a state machine. Hanh eta al. proposed a set of mutation operators for simulation models build with MATLAB/Simulink tool [HBT16].

To evaluate the mutation score of the seeded fault, the original model and the mutated model (i.e., mutant) are connected to the test harness containing the test cases. In the field of simulation-model based testing, a test case consists of a number of signals that stimulates the SUT (i.e., both the non-mutated and the mutated models). Thus, same test inputs (i.e., the same signal test data values) are placed at the input ports of both models. The out ports of each model, provide the resulting signals of the execution of the test after the established simulation time is elapsed. A special block can be modeled as *oracle* to gather resulting out port groups of signals from both models to compare them and provide a verdict. When certain mutant out port value differs from equivalent non-mutated out port value, the mutant has been *killed*, detecting the fault.

The mutation testing technique has been applied in both the exploratory contributions described in Chapters 5 and 6 as well as in the main contribution presented in Chapter 7. The technique has been employed to assess the approaches proposed in the contributions.

2.4.3 Quality metrics for testing HCCPSs

Cost-effective techniques are required for testing configurable simulation models, as testing all possible combinations is unfeasible [AWA+18, MNBB19]. This section defines both black-box and white-box test quality metrics for configurable simulation models relying on 150% variability modeling approaches.[2] The defined test quality metrics can assess the effectiveness of a test case or a set of test cases when testing configurable simulation models. The section is divided into white-box and black-box test quality metrics and equations are adapted to the context of configurable simulation models.

White-box metrics

White-box testing focuses on techniques that rely on SUT internals (e.g., internal structure of the system or dependency of the data on the execution paths) [AO16]. The number of test objectives in a system can depend on the type of coverage it is aimed at to be employed (e.g., data-flow or control-flow) as well as the metrics (e.g., Condition Coverage or Decision Coverage). Consider Table 2.2 as an example, which presents the relation among 4 features (i.e., f_1, f_2, f_3, f_4) along with eight different test objectives (to) with nine different test cases (i.e., TC1 to TC9). Let us suppose that product P_k is obtained after deriving the product line configured with two features $P_k = \{f_1, f_2\}$. Therefore, product P_k contains five test objectives $TO_{P_k} = \{to_1, to_2, to_3, to_4, to_5\}$ associated to product assets. Let us suppose also a test suite composed of the test cases TC1 and TC2 that exercise the P_k product $TS_{P_k} = \{TC1, TC2\}$.

- **Structural coverage**

  Structural coverage is a measure widely used as a quality metric [YH12], moreover, in recent decades, this measure has also been adopted by model-based engineering [UPL12]. The purpose of the structural coverage is to determine the amount of code (or model) that has been exercised during a testing activity. There are multiple control-flow criteria for structural coverage [AO08]. Three of the most common criteria are: Decision Coverage (DC), Condition Coverage (CC) and Modified Condition/Decision Coverage (MC/DC) [UL10]. The DC criterion checks the outcome of a decision. To this end, a minimum of two test cases are generated: one for a true outcome of the decision and another one for a false one. The CC criterion focuses on

[2] Refer to Section 4.4.1 for detailed explanation of 150% Models in the context of configurable simulation models.

Table 2.2: Test objectives, features and test cases exposed.

	f1		f2			f3		f4
	to1	to2	to3	to4	to5	to6	to7	to8
TC1	X		X		X	X		
TC2		X		X		X	X	
TC3	X	X			X			X
TC4	X	X	X			X		X
TC5			X	X				X
TC6	X		X				X	
TC8		X		X				
TC9				X	X		X	X

all possible values that every condition of the decision can take, irrespectively of the decision outcome. The MC/DC criterion examines that (i) every possible outcome for each decision is checked, (ii) every possible outcome for each condition is checked and (iii) each condition in a decision is shown to independently affect the outcome of the decision [HVCR01]. Refer to Appendix A where we have introduced structural coverage principals for further details.

In our context, the coverage is measured in terms of covered test objectives that are associated with product line assets. When specific product configurations are derived from the product line, a particular number of test objectives are implicitly derived. The number of test objectives in the specific product is equal to or less than the number of test objectives of the entire product line. The structural coverage can be measured in each configuration at the application engineering level as well as at the Domain Engineering level. Refer to Chapter 6 to have a detailed description of how to measure the structural coverage at the application and domain engineering level. Considering the example of Table 2.2 the structural coverage obtained by the product at the application engineering level (i.e., at the product level) will consist of the number of test objectives covered in relation to the number of test objectives in the product. In this case, the five test objectives of the product will be covered (i.e., 100% structural coverage of the product will be achieved). However, the structural coverage obtained by the product with the same test suite at the domain engineering level will consist of the number of test objectives covered relative to the total number of objectives of the entire product line. In this case, the number of covered test objectives

is the same (i.e., 5 test objectives) but there are 8 test objectives in the entire product line, so the achieved structural coverage at the application engineering level would be $5/8 = 62\%$.

- **Feature coverage**

  This metric is inspired in the Feature Coverage presented by Wang et al. [WAGL16a], however it is instrumented through test objectives and adapted to configurable simulation models context. Test objectives are associated to product line assets, and these assets are associated to features. Bearing this in mind, it is possible to obtain the extent to which a feature has been tested in a simulation model. Let us suppose that $TO_{f_i} = \{to_1, to_2, ..., to_{N_{f_i}}\}$ are the test objectives covering feature i, and these test objectives are part of the entire product line, and thus can appear in any configuration (i.e., $to_a \in TO_{150\%}$). Intuitively, the coverage for a given feature is the percentage of its test objectives that have been covered (i.e., $\sum_{i=0}^{N} to_i/N_{f_i}$). Considering again the example in Table 2.2, the first test case (i.e., $TC1$) would obtain a test coverage for the first feature of 50%, as it only covers $to1$, 66.66% for the second feature as it covers two test objectives out of three, 50% for the third feature and 0% of feature coverage for the fourth feature. If we complement the first test case with the second, we have a 100% of feature coverage for $f1$, $f2$ and $f3$, since $TC2$ covers those test objectives not covered by $TC1$ for these first three features. However, the feature coverage for $f4$ remains at 0%. To obtain a full feature coverage, we need to complement these two test cases with one of the test cases covering to_8, which is the test objective related to $f4$; thus, either $TC3$, $TC4$, $TC5$ or $TC9$ can be selected. The feature coverage of the entire product line will be the sum representation of the feature coverage obtained by each feature (i.e., $\sum_{i=0}^{N} cov(f_i)/Nf$, being $cov(f_i)$ the coverage obtained for feature i and Nf the total number of features in the product line).

- **Feature pairwise coverage**

  It is well known that many faults in product line engineering appear due to the interaction of pairs of features [CDS08]. Similarly to the feature coverage metric, this one is also inspired in the one proposed by Wang et al. [WAGL16a] with equal adaptations. In a simulation model, this could be measured by considering the interaction of test objectives when these are associated to different features. When considering Table 2.2, 6 feature pair interactions exist (i.e., f1-f2, f1-f3, f1-f4, f2-f3, f2-f4, f3-f4). If we consider the coverage for the first interaction (i.e., interaction of f1-f2), the interaction

between the test objectives of these features would need to be considered (i.e., to1-to3, to1-to4, to1-to5, to2-to3, to2-to4 and to2-to5). Note that in this case, selecting TC1, two of the objective interactions have been covered, and thus, a 33.33% of feature pairwise coverage is scored for the interaction of f1 and f2. Similar to the feature coverage, the feature pairwise coverage of the entire product line is calculated as the sum representation of the coverage obtained by each feature pair (i.e., $\sum_{i=0}^{N}(\ \sum_{j=i+1}^{N}\ fpcov(f_i, f_j))/Nfpair$, being $fpcov(f_i, f_j)$ the feature pair coverage for features i and j and $Nfpair$ the total number of feature pairs in the product line).

Black-box metrics

Black-box testing focuses on techniques that rely on external information of the SUT (e.g., specifications, requirements, design, etc.) [AO16]. For the particular field of testing simulation models with black-box techniques, Matinnejad et al. [MNB17, MNBB19] proposed different black-box metrics based on two factors: (i) on the one hand the anti-patterns of simulations models, and (ii) on the other hand a similarity metric based on the euclidean distances of inputs and outputs. Arrieta et al. [AWA+18] adapted these metrics for the test case selection context in the field of simulation models. We later proposed these metrics for the context of configurable simulation models.

- **Anti-patterns**

  Three different anti-patterns were presented by Matinnejad et al. in [MNB17], which can be seen in Figure 2.10. Instability aims at measuring quick and frequent oscillations of a signal [MNB17]. Discontinuity is an anti-pattern in which an output signal shows a short duration pulse [MNB17]. Lastly, growth to infinity is an anti-pattern where an output signal shows how a signal grows to an infinite value [MNB17]. Arrieta et al. [AWA+18], adapted those anti-patterns proposed in [MNB17, MNBB19] to be normalized independently of the number of outputs a model had and the data-type and maximum/minimum values of their outputs. Refer to [AWA+18] for further detail of how each anti-pattern is measured for each signal. For configurable simulation models, the anti-pattern degree for each of the test cases given a test suite (i.e., TS) is measured by observing each output of the simulation model. However, since each output in O_{150} can represent a particular variable in different units (e.g., one output can represent vehicle speed in km/h while another output can represent the acceleration in m/s^2), it is

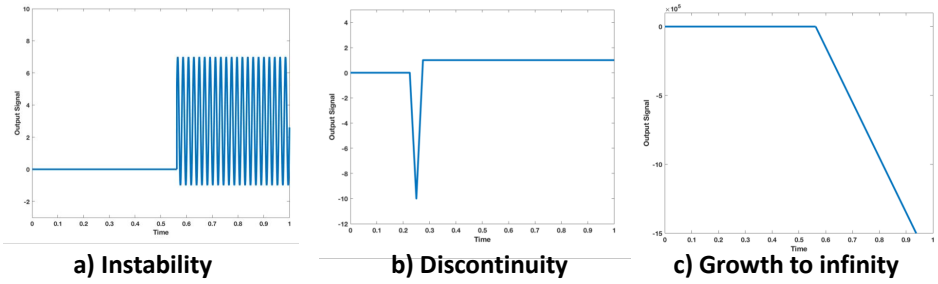


Figure 2.10: Anti-patterns for simulation models proposed by Matinnejad et al. [MNB17]

mandatory to normalize the anti-pattern degree of each test case. Given a 150% simulation model with N_{out} outputs (i.e., $O = \{o_1, o_2, ..., o_{N_{out}}\}$), the anti-pattern degree of a test case j in TS is obtained with Equation 2.1, where $ap(Osig_{j_i})$ is the anti-pattern degree of the i-th signal for tc_j and $\max(ap(Osig_i))$ is the maximum anti-pattern degree obtained in the i-th signal when considering all test cases in TS. Note that ap can be either instability, discontinuity or growth to infinity.

$$TCap(tc_j) = \frac{1}{\max(ap(Osig_i))) \cdot M} \times \sum_{i=1}^{M}(ap(Osig_{j_i}) \tag{2.1}$$

■ **Similarity**

It is well known that a set of different test cases is more likely to detect faults [FPCY16, HAB13]. By following this principle we have defined two similarity measures for configurable systems. Firstly, we have defined the similarity between test cases adapting the Euclidean distance measure proposed for simulation models [MNB17, MNBB19, AWA+18]. To measure the similarity of test cases we employed the Euclidean distance, which is the similarity measure proposed for simulation models in previous studies [MNBB15, MNBB19]. Nevertheless, unlike these previous studies, in the context of this study, the Test Execution Time (TET) of test cases might differ, which means that the input and output signals of two test cases can have different numbers of simulation steps. Subsequently, we have adapted the Euclidean distance between test cases to deal with this problem. Given two signals related to a specific input or output (i.e., sig and sig') in two different test cases, the Euclidean distance is measured following Equation 2.2. Given two different signals, $\min(k_{sig}, k_{sig'})$ is the number of steps of the signal whose test case has a lower TET, $max_{R_{sig}}$ is the maximum value that the signal in

the simulation can obtain, $min_{R_{sig}}$ is the minimum value that the signal in the simulation can obtain and K is the number of steps that the test case with the highest TET in the test suite has. We consider K as the number of steps for the longest test case for two main reasons. First, to ensure that the distances between test cases are normalized. Secondly, because a longer test case might have a higher chance to detect faults, and thus, we penalize the distance of those very short test cases. We remark that in our study the simulation step (Δt) is the same for all test cases. When considering Equation 2.2, a higher distance means that two signals are more dissimilar.

$$Dist(sig, sig') = \frac{\sqrt{\sum_{i=0}^{\min(k_{sig}, k_{sig'})} (sig(i \cdot \Delta t) - sig'(i \cdot \Delta t))^2}}{\sqrt{K+1} \times (max_{R_{sig}} - min_{R_{sig}})} \quad (2.2)$$

Equation 2.3 defines the input signal-based distance between two different test cases (TC_a and TC_b) for a simulation model with N inputs (in the 150% simulation model). When comparing two test cases executed in two product configurations, three group of signals must be considered in simulation models: (i) the signals that both configurations share, (ii) the signals that are present only in one configuration (but not in the other) and (iii) the signals that both configurations do not have. Equation 2.3 details how to measure the distance between two test cases (i.e., TC_a and TC_b). $D(sig_{a_i}, sig_{b_i})$ is referred to the Euclidean distance between the signal i of test case a and test case b, where S is the number of signals that both configuration models share (i.e., the first group). We consider that a distance between two signals is the maximum when a configuration has one specific signal and the other one does not have it. The maximum distance ($maxD$) will be 1, and d will be the number of signals that a specific configuration has but the other configuration doesn't. Lastly, if both configurations do not have a specific signal, we consider that from the point of view of the test case similarity, both are the same, and we consider the minimum distance (which is 0 and thus, not considered in the equation).

$$TCD(TC_a, TC_b) = \frac{\sum_{i=1}^{s} Dist(sig_{a_i}, sig_{b_i}) + \sum_{i=1}^{d} maxD}{N} \quad (2.3)$$

The previous equation can also be complemented with the similarity that two independent configurations have in terms of features (e.g., by considering the similarity measure proposed by Al-Hajjaji et al. [AHTL+16]).

$$Sim(P_u, TC_a, P_v, TC_b) = PD(P_u, P_v) \cdot Wp + TCD(TC_a, TC_b) \cdot Wtc \quad (2.4)$$

For instance, given a configuration u (C_u) and a configuration v (C_v), and two test cases (e.g., TC_a and TC_b), where TC_a is tested on C_u and TC_b on C_v, the distance between both test cases and configurations can be measured as a weighted sum of both. Where the distance between both test cases (i.e., $TCD(TC_a, TC_b)$) is calculated following for instance Equation 2.3 and the distance between both products (i.e., $PD(P_u, P_v)$ as given in Equation 2.4. Note that Wp is the weight given to the distance between both products whereas Wtc is the weight given to the distance between test cases.

This collection of metrics and adaptation to the context of configurable simulation models was published in the *REVE 2019: Seventh International Workshop on Reverse Variability Engineering In Proceedings of the 23rd International Systems and Software Product Line Conference* [MAES19b]. White-box structural-coverage based metrics were previously published by the author in [MAES19a], while white-box feature related metrics presented in this section are authored by [WAGL16a]. Although the metrics were already intended for SPLs, we adapted the metrics to measure test objectives in the context of the configurable simulation models. Black-box anti-pattern and similarity metrics presented in this section are authored by Matinnejad et al. [MNBB19]. These metrics were adapted by Arrieta et al. [AWA+18] for the test case selection context for single simulation models. We adapted these black-box metrics to the configurable simulation models field. We have considered this work as a minor contribution and have therefore included the metrics as basic concepts in this background section.
White-box structural coverage metrics are employed to improve test case selection in the exploratory contribution described in Chapters 6 and black-box similarity metrics are employed for test case prioritization in the main contribution presented in Chapter 7.

Chapter 3

State of the Art

Contents

This chapter presents the state of the art of HCCPS test optimization. As described in Section 2.3.3 the optimization of HCCPSs rely on testing strategies of the SPL field. Thus, we first present a review of SPL testing strategies and set the contextual framework in which the thesis operates introducing the SPL test optimization reference process. The chapter is subsequently structured by reviewing the steps of the introduced reference process. Finally, the critical analysis of the state of the art is presented.

3.1 Testing Strategies

In this thesis we have revised the literature regarding SPL testing strategies. We have mainly relied on systematic literature reviews, systematic literature mappings and surveys. Selected 21 works are presented grouped chronologically in five stages.

A first group of works from beginning of the 2000s analyzed the techniques and activities to address testing in SPLs [McG01, KM03]. McGregor [McG01] described the test-related techniques and activities to be used to form the test process for a product line. Kolb and Muthing [KM03] discussed in their publication the challenges of testing SPL as opposed to testing individual products.

A second group of early works provided three general SPL testing strategy classifications [TTK04, PBL05, RRKP06]. Tevanlinna et al. [TTK04] carried out a survey on SPL testing. They categorized approaches into four different strategies for SPL testing: (i) the *Product by product* strategy, that relies to the application engineer who tests each product individually without reusing the test assets. (ii) In the *Incremental testing* strategy first product is tested individually while the following products are partially tested using regression testing. (iii) In the *Reusable asset instantiation* strategy abstract and specific test assets are created in DE. Later in AE test assets are instantiated, where specific test assets are use as is and abstract test assets are extended to the product-specific requirements. (iv) The *Division of responsibilities* strategy divides the testing activity of the V-model levels between the different engineering units. Reuys et al. published a book chapter where their proposal of ScenTED Method was described [RRKP06]. They identified three SPL testing strategies: (i) the *Separate test case development* strategy, develops independent test cases for each derived application (i.e., no reuse is performed). (ii) The *Opportunistic reuse of existing test cases* strategy, develops test cases

for first derived application and reuse them as soon as new applications are derived from product line. (iii) The *Design test cases for reuse* strategy proposes to create reusable test artifacts in DT. Later in AT, test cases for specific application are derived from the reusable test artifacts.

In the context of the published book describing the SPLE framework [PBL05], Pohl laid the groundwork for what would become the foundations of the product-line community's activity in the years to come, including four SPLs testing strategies. (i) The *Brute Force Strategy (BFS)* performs all test activities at all test levels and for all possible applications during DT. (ii) The *Pure Application Strategy (PAS)* perform tests only in AE, for which only application-specific tests are created (i.e., no reusable artifacts are employed). (iii) The *Sample Application Strategy (SAS)* uses one or several sample applications, which are tested at DE. Later when specific products are derived in AE, specific testing is also required for products. (iv) The *Commonality and Reuse Strategy (CRS)* tests common parts in DT and prepares reusable test artifacts for variable parts. Later in the AT the predefined and variable DT artifacts are employed and adapted to test specific applications.

In a third stage of publications, different systematic mappings, reviews and surveys were conducted but no new general testing strategies were defined [LUV09, ER11, JHF11b, NdCMM$^+$11]. Lamancha et al. [LUV09] conducted a systematic literature review, where they analyzed 37 papers. The contributions and approaches were classified according to 7 categories: unit testing, integration testing, functional testing, SPL architecture testing, embedded systems testing, testing process, testing effort in SPL y test product generation. Engström et al. [ER11] conducted a systematic mapping study reviewing 64 papers to analyze approaches, publication fora and remaining challenges. They highlighted main challenges in SPL testing as: (i) the large number of tests, (ii) balance between test effort for reusable components and concrete products, and (iii) handling variability in testing. Furthermore, they concluded the need for more empirical studies. Johansen et al. [JHF11b] conducted a survey of three empirical case studies. They concluded that the reported improvements corresponded to the reusable component testing strategy. They also identified as a future improvement the need to study interaction failures between components. Neto et al. [NdCMM$^+$11] conducted a systematic mapping study in which nine key (at the time) research questions were comprehensively analyzed. From the study we highlight the review of testing strategies for SPL, which compiled the proposals of Tevanlinna[TTK04] and Reuys[RRKP06] into a unified list

of 5 strategies: Testing product by product, Incremental testing of product lines, Opportunistic reuse of test assets, Design test assets for reuse, Division of responsibilities. We do not reproduce what each of the strategies consists of as these have already been described by the primary authors.

In a fourth stage of publications, testing strategies defined by Pohl [PBL05], Tevanlinna [TTK04] and Reuys [RRKP06] were employed when performing surveys or published book chapters[OWES11, RE12, LKL12, TAK+14, dCMMCDA14]. Oster et al. published a book chapter [OWES11] with a survey of model-based SPL testing, where six approaches including their methodology based on MBT paradigm were studied and compared through a common running example. In the introduction of the survey the four SPL testing strategies defined by Tevanlinna [TTK04] are described as they are well suited to Model-Based testing methodologies. Refer to the primary paper for details of the strategies. Runeson and Engström published a book chapter [RE12] where they analyzed the state of the art of regression testing for SPLs with the aim of translating the benefits of regression testing to the context of SPLs. The paper identifies approaches based on formal modeling of variants and versions for SPLs as well as a visualization-based approach. When classifying the approaches regarding SPL testing strategies, they rely on the classifications made by Pohl [PBL05] and Tevanlinna [TTK04]. Refer to the primary paper for details of the strategies. Lee et al. [LKL12] conducted a (non-systematic) survey that analyzed 15 selected papers. For the study they relied on the W-shaped SPL testing reference process (described by [JHQJ08])). They posed 8 research questions that were used to analyze the selected papers. The survey does not perform an analysis of possible testing strategies, focusing all the study on the W-shaped reference process. Thum et al. [TAK+14] carried out a survey of 123 works, to classify existing static analysis (not testing) approaches. They classified existing works into three main analysis strategies (product-based, featured-based and family-based) and four extra strategies resulting from combinations of previous ones. The classification was performed according to how the analysis was handling the variability in each strategy. They compared this static analysis strategies with the ones proposed by Pohl [PBL05]. In [dCMMCDA14] doCarmoMachado et al. conducted a systematic literature review of 49 works covering the period from 1998 to 2013. Two interests were reported to be taken into account when approaching SPLs testing strategies: (i) The *need to handle the selection of products to test* to reduce the set of possibilities to a reasonable and representative set of

products configurations. (ii) The *necessity to handle the test of end-product functionalities*, i.e., how to handle testing of selected products with regard to variability definition, assets reuse, test automation and binding to Domain or Application level.

A fifth stage of publications, as the community developed the field with new proposals and approaches, new studies and surveys were conducted to analyze the status of applying Combinatorial Interaction Testing (CIT)[1] or Evolutionary Computation techniques in SPL testing as well as to review the notion of SPL test coverage criteria [LHFRE15, LHFC$^+$16, FLHE18, VAHT$^+$18, LKJ20]. Lopez-Herrejon et al. [LHFRE15] conducted a systematic mapping study where 47 works proposing approaches that employed CIT for testing SPL purposes were comprehensively studied. They analyzed four aspects: CIT techniques, CIT phases, case studies and publication fora. From the work they concluded that among the 13 techniques employed in CIT, Greedy algorithm-based techniques along with SBSE-based techniques were the most numerous. They also concluded that CIT is mainly applied to obtain a representative subset of products from the product line. They also highlighted the need for a community wide benchmark. Varshosaz et al. conducted a meticulous classification of product sampling for SPLs [VAHT$^+$18]. 48 works were surveyed (without systematic process) and criteria for classification defined. They concluded that the majority of techniques only used feature models as inputs but other input types were understudied. Regarding techniques, greedy and meta-heuristic were the most common ones while coverage and diversity based techniques were identified as opportunities. Lopez-Herrejon et al. published a book chapter [LHFC$^+$16] compiling the overview and challenges of Evolutionary Computation in the context of SPL testing. The work documents the uses of SBSE in SPL testing and also points out several challenges for the community to solve. Among them, we highlight the potential benefit of employing multi-objective optimization, the untapped potential of exploiting the DE level knowledge of SPLs or further test suite prioritization opportunities. Fischer et al. [FLHE18] created a benchmark for evaluating the fault detection capabilities of SPL testing approaches. To this end, they engineered a process to automatically generate tests and failures. They integrated and adapted tools from third parties into their process (i.e., *EvoSuite* for test case generation and µ*Java* for seeding mutants). Result and limitations were presented along with future lines of work. Lee et al. [LKJ20] conducted a systematic literature review

[1] Refer to Section 3.2 for detailed explanation of the CIT concept.

thoroughly analyzing 78 works. Selected works were classified according to dimensions: (i) employed test strategy for which Polh's classification was employed [PBL05] and (ii) specific test coverage criteria to each SPL testing method. With the exception of test coverage criteria of product selection, existing SPL test coverage mainly correspond to test coverage criteria for single products. Consequently, they emphasized the importance of clarifying and defining the notion of SPL test coverage criteria in future works.

To conclude the analysis of the selected works, we have identified three primary classifications of SPL testing strategies. These primary works are Tevanlinna [TTK04], Pohl [PBL05] and Reuys [RRKP06] as classifications proposed by them have been repeatedly referenced in the other analyzed works. Table 3.1 compares the three classifications with each other in order to clarify the terminology used by the different works. As a result, up to 7 different strategies are proposed when classifications are unified.

According to the authors of the classifications studied, some strategies are unfeasible (i.e., *Brute Force Strategy*), while other strategies are not recommended for product lines that can generate a large number of products (i.e., *Pure Application Strategy*). Both *Incremental testing of product lines* and *Opportunistic reuse of existing test cases* strategies are based on testing an initial product (in AE) to reuse generated test artifacts when testing subsequent products (applying regression testing techniques in the particular case of *Incremental testing of product lines*). The option that enables early validation is the *Sample Application Strategy*. In the classification, *Commonality and Reuse Strategy* best addresses the design for reuse principle. The proposal to balance both early validation and reuse is to develop variable artifacts (as in CRS) but creating meaningful product parts in DE that can enhance early validation (similar to SAS). Finaly, the *Division of responsibilities* structures testing according to V-Model levels divided in different engineering units.

In this thesis we have addressed the problem of test optimization in the context of the SAS strategy. Considering the work of CarmoMachado [dCMMCDA14] where it was highlighted that there are two (independent but complementary) interests that an SPL testing strategy must address: (i) product selection and (ii) management of final product testing. According to these two interests, we have organized the review of the state of the art based on *optimization at the product level* and *optimization at the test case level*.

Table 3.1: Comparison of SPL testing strategies classifications. First column corresponds to classification provided by Tevanlinna et al. [TTK04], second column corresponds to Paul et al. [PBL05] and third column corresponds to Reuys et al. [RRKP06]. Strategies from different classifications presented in the same line are considered nearly equivalent.

Tevanlinna et al. [TTK04]	**Pohl et al.** [PBL05]	**Reuys et al.** [RRKP06]
	Brute Force Strategy All test activities are performed at all test levels and for all possible applications during Domain Testing.	
Product by Product Delivered products are fully tested one by one in Application Engineering.	**Pure Application Strategy** Tests are performed only in Application Engineering. Only application specific tests are created and performed. No reusable Domain test artifacts are created during Domain Testing.	**Separate test cases development** Test cases for each derived application are developed independently (without reuse).
Incremental testing of product families First derived product is tested individually and following products are tested using regression testing techniques. It is performed at Application Engineering but taking advantage of commonalities.		**Opportunistic reuse of existing test cases** Test cases are developed for first application derived from product line. Next derived applications reuse (if possible) previously generated test cases.
	Sample Application Strategy One or a few sample applications are used to test the Domain artifacts. Application Testing is still required for each application.	
Reusable asset instantiation Test assets are created in Domain Engineering (as extensively as possible), including variable assets (abstract test cases). In Application Engineering a full testing process is instantiated (concrete test cases are reused and abstract test cases extended to product-specifities).	**Commonality and Reuse Strategy** Domain Testing aims at testing common parts and preparing test artifacts for variable parts. Application testing aims at reusing the test artifacts for common parts and reusing the predefined, variable Domain test artifacts to test specific applications.	**Design test cases for reuse** In Domain Testing reusable test artifacts are created, whereas in Application Testing test cases for a specific Application are derived.
	Combined SAS/CRS Reusable test artifacts are created in Domain Testing and the reuse of these artifacts are ensured in Application Testing. In addition, an early validation is performed with fragments of a sample application. No complete application is built, but only parts that are large enough to perform the tests.	
Division of responsibilities Testing is structured in the levels of the V-Model divided in different engineering units.		

3.1.1 SPL test optimization reference process

Taking as input the testing optimization approaches both at product level and test case level, we have modeled the SPL testing in a reference process composed of five-step. A SPEM-based diagram depicts in Figure 3.1 the reference process where the steps have been divided according to the two interests described in [dCMMCDA14] (i.e., optimization performed at the product level and optimization at the test case level).[2]

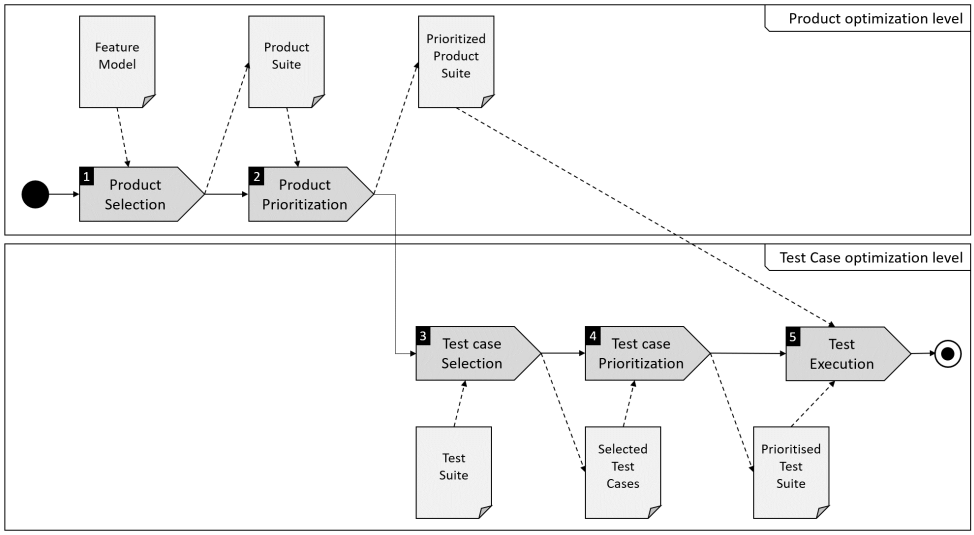


Figure 3.1: SPL test optimization reference process.

The first step (1), titled *Product Selection* in the diagram, takes the variability model (e.g., usually a *Feature Model*) as input. Then, it derives the product set to be tested, titled *Product Suite*, this is usually done by applying combinatorial testing algorithms which satisfy the selection criteria. Secondly (2), at the *Product Prioritization* step, products of the product suite are sorted, scheduling at the beginning those ones which fit better the defined objective (i.e., better fault detection rate). As a result, the *Prioritized Product Suite* is obtained. At the third step (3), titled *Test case Selection*, relevant test cases that must be executed in each of the products are selected and minimized To this end, the *Test Suite* is taken as input and the test cases are selected according to the features of each product to be tested (e.g., only test cases that can be applied to the product are selected). In addition, minimization criteria can be applied to the selected test cases (e.g. discard low fault detection rate

[2]The Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0 was employed to create the figure. http://www.omg.org/spec/SPEM/2.0/

test cases or duplicated test cases) in order to optimize the number of test cases to be applied to each product. As output, the *Selected test cases* are obtained. The fourth step (4) consist of prioritizing the test cases, for which the selected (and minimized) test cases are taken and sorted according to a criterion. As a result the *Prioritized Test Suite* is obtained. Finally, at the fifth step (5), following the defined product order in the *Prioritized Product Suite*, each of the products is thoroughly tested by the test cases, applying the order defined at the *Prioritized Test Suite*.

The remaining part of the chapter reviews the state of the art of the first four steps described in the SPL test optimization process and related critical analysis.

3.2 Product Selection

The high number of feature combinations in a SPL produces a explosion of possible products to be tested. To overcome this problem, product selection aims to reduce the number of products to be tested. In relevant systematic studies [LHFRE15, dCMMCDA14, dMSNdCMM+11, ER11, LUV09, TAK+14] CIT is identified as the leading selection approach for testing in SPL. The CIT approach is based on the observation that most of the defects are expected to be caused by an interaction of few features [KAuR+09]. Thus, CIT selects a reduced number of products where interaction errors are more likely to occur [CDS08]. The selection is achieved choosing the products of the SPL to cover all interactions between t features, known as *t-wise*. For example, in 2-wise (A.K.A. pairwise) all possible interactions between feature pairs are chosen to be tested. Leading works indicate [dCMMCDA14, LHFRE15] that for selecting a subset of representative products from the product line, CIT is *de facto* the standard technique employed by the community. Moreover, for solving problems of high complexity (where we can locate the product selection step), the heuristic reduction provided by pairwise can provide a practical mechanism for representative (product line) sampling. This approach has been widely adopted by the research and industry community [TAK+14, dCMMCDA14] and several algorithms have been proposed to perform the *t-wise* interactions (e.g., ICPL, AETG, CASA, Chvatal, etc.).

SPL variability generates strong constraints between features, making *t-wise* testing more difficult in large SPLs. Several works have been published in order to solve this problem. Cohen et al. [CDS07] analyzed the interaction

testing taking into account constraints and proposes techniques to handle them and integrate with CIT tools. Perrouin et al. [PSK+10] proposed an approach for scalable tool set using Alloy to automatically generate test cases satisfying *t-wise*. Oster et al. [OMR10] proposed an approach and a Framework named MoSoPoLiTe where pairwise combinatorial is performed, with graph transformation and forward checking. Johansen et al. [JHF11a] proposed the SPLCAT approach, where covering arrays [CDS07] are generated. Nevertheless, the approach is limited to 3-wise testing.

While CIT focuses on reducing the products to be tested, the approach proposed by Henard et al. [HPP+13a] focuses on selecting products to optimize fault detection. Thus, Henard et al. [HPP+13a] proposed an approach with two mutation operators to assess the ability of test suites to detect errors when deriving products from feature models. Results demonstrate that dissimilar product suites have higher mutant detection ability, validating the similarity-driven product line testing. Additionally, Henard et al. [HPP+13b] also proposed the use of constraint solving technique to prune invalid products from the search space and a genetic algorithm approach to generate products handling multiple objectives. Furthermore, Henard et al. [HPLT14] introduced a search-based approach to generate sets of products focusing on possible faulty implementation of the FM that should be tested.

In [GCD11] Garvin et al. search-based heuristics were employed improving an extension to the AETG algorithm [CDFP97] using simulated annealing. Ensan et al. [EBG12] proposed evolutionary testing approach GA, searching in the configuration space of a software product line feature model in order to automatically generate product suites.

3.3 Product Prioritization

While most of the approaches focus on reducing the number of products to be generated and tested, several authors have worked on prioritizing generated products as was reported in surveys and mapping studies [NL11, LHFC+16].

Initial product prioritization propositions [BC07, YHTS09, BSL+10, UKB10] opened the way to the latest research works [SSRC14a, AHTM+14]. In [BC07] Bryce et al. proposed search-based techniques to prioritize products in terms of t-wise coverage. In [YHTS09] Yoo and Harman proposed a cluster-based prioritization to reduce the number of pairwise interactions. [BSL+10] incorporated the test-cost metric to the CIT coverage metric.

Al-Hajjaji et al. [AHTM+14] proposed a similarity-based approach to prioritize the products to be generated (before generating products). The similarity of products was calculated based on Hamming distance [Ham50]. The algorithm was implemented and integrated into the *FeatureIDE* tool [TKB+14], in order to automatize the prioritized product suite generation. Results showed that similarity-based product prioritization is more effective than default product ordering provided by the product selection.

Similarly, Sanchez et al. [SSRC14a] explored the product prioritization techniques in SPL testing for already derived product suites.[3] Five different prioritization criteria based on common metrics of FM were proposed, comparing their fault detection effectiveness. Three of these criteria were related to complexity, another criterion was related to reusability and last criterion was related to the dissimilarity of products. Results showed that product prioritization accelerates fault detection of product suites generated both with CIT techniques or randomly. Moreover, Sanchez et al. [SSRC14b, SSPRC15] performed a thorough work with the Drupal framework, where a real HCS was analyzed. Obtained results, highlighted that the size of the feature, the cyclomatic complexity, number of changes and fault history are correlated with the number of bugs. Additionally, it was demonstrated that working with non-functional attributes, such as number of reported installations, number of developers or number of changes among others, accelerates fault detection, outperforming the results obtained with similarity-based product prioritization.

Finally, an extended work was published by Parejo et al. [PSS+16] focusing on a multi-objective approach of the product prioritization. To this end the NSGA-II evolutionary algorithm was adapted and the Drupal case study serve as test bench. Results revealed that prioritization driven by non-functional objectives accelerate the detection of bugs more effectively than using prioritization driven by functional objectives. Additionally, results showed that prioritization objective based on pairwise coverage combined with other objectives is effective detecting bugs quickly.

3.4 Test Case Selection

Lopez-Herrejon et al. [LHLE15] indicated that SPL testing is the most prevalent application of SBSE. However, most of the work was focused on DE testing

[3] DISAMBIGUATION: Note that some authors in the literature (such as [SSRC14a, BC07]) refer to the product selection problem as test case selection.

activities, with a minimal impact in AE testing activities. The reasoning performed by Lopez-Herrejon et al. [LHLE15] concluded two observations: (i) The large number of individuals required by SBSE techniques is naturally more profitable at the DE level. (ii) The second observation brought a conclusion obtained by Metzger et al. [MP14] who pointed that AE testing research activities have focused mostly on deriving test from reusable artifacts or minimizing the retesting of parts already tested.

A challenging work line was opened by Wang et al. [WAG15] with the proposed research work to select and minimize test cases. For test case selection, an initial methodology was proposed [WGAL13] for automated selection of test cases for a new product using FM, Component Family Model (CFM) [4] and a test case repository. Traceability links were established between CFM and the test case repository, and also between FM and CFM. To test new products, the engineer selects features from the FM and corresponding test cases can be automatically obtained from the test case repository. The approach has been successfully corroborated with industrial studies [WAGL15, WAGL16b].

Arrieta et al. [AWSE16a] proposed a search-based test case selection approach adapted to the "X-in-the-loop" test levels of CPS product lines context. The approach proposes a process of four steps: (i) in the first step the variability modeling is designed using FM to manage variability and requirements. (ii) In the second step configuration selection is performed. First and second steps are conducted with the FeatureIDE tool [TKB+14]. (iii) In the third step test cases are selected employing search-based algorithms. FM and configuration selection information is used to identify selected requirements. Subsequently, search algorithms are employed to select test cases at MIL, SIL and HIL levels. (iv) Finally, in the fourth step, test cases are executed using Simulink tool. Results showed that among the three search algorithms employed in the empirical evaluation (i.e., GA, Greedy, Alternating Variable Method), GA results the most appropriate algorithm for solving test case selection.

However, even after test case selection is performed the number of test cases to execute results may be unfeasible. To overcome this problem Wang et al. [WAG15] proposed an test suite minimization approach to identify and eliminate redundant test cases. Moreover, to avoid loosing effectiveness of various optimization objectives (i.e., test coverage, fault detection capability or time budget) when minimizing, the approach was outlined as a search problem.

[4] A Component Family Model describes the internal structure of the individual components of a product line and their dependencies on the features [ps06].

To deeply research best fitness function for this end, Wang et al. [WAG15] conducted the empirical evaluation of an industrial case study with three weight-based GAs and seven multi-objective search algorithms. Results showed that Random-Weight Genetic Algorithm outperforms the other algorithms. Finally, a TEMSA named tool was implemented in order to support test minimization in the context of SPL.

3.5 Test Case Prioritization

HCSs present a double challenge when addressing testing. The first challenge is the aforementioned elevated number of test cases to be executed and the second challenge is related to the limited testing budget available. Thus, often it is not possible executing all test cases despite the test case selection and minimization effort. Consequently, best test cases must be prioritized in order to guarantee that test cases that fit best to testing objectives and budget are executed first.

Wang et al. [WBA+14] proposed a search-based technique for cost-effective test cases prioritization for a given limited budget addressing to multiple cost and effectiveness objectives. To this end three search algorithms (i.e., Alternating Variable Method (AVM), GA, (1 + 1) Evolutionary Algorithm ((1+1)EA)) were empirically evaluated within an industrial case study with 500 designed artificial problems. Results showed that (1+1)EA achieved the significant best performance among the selected search algorithms when finding an optimal solution for the test prioritization problem. The evaluation also showed that algorithms are not influenced by the increasing complexity of problems, ensuring the scalability of search algorithms.

Arrieta et al. [AWSE16b] proposed an approach based on weight-based search algorithms for prioritizing the test cases for configurable CPSs. The approach employs Fault Detection Capability (FDC) obtained from historical data as an effectiveness measure and test execution time as cost measure. Two case studies were used to evaluate the performance of the studied algorithms (i.e., weight-based genetic algorithm, random weighted genetic algorithm, Greedy and AVM). The assessment of the effectiveness of the proposed approach was performed with mutation testing. Results reflect that local search algorithms (i.e.,Greedy and AVM) showed best performance solving the search problem.

3.6 Critical analysis of the state of the art

SPLs have been intensively researched [HPMFA+16, LHLE15, MP14, TAK+14, BSC10] and applied [Wei08, BBSR16] in several disciplines such as embedded systems for automotive, avionics or medical devices [LSR07]. However, SPL testing must still face several challenges as it has been pointed in different reviews [MP14, LHFC+16]. The following sections analyses the progress of the reviewed field in the state of the art.

3.6.1 Product selection and prioritization

Section 3.2 presents a summary of **product selection**. As Lopez-Herrejon et al. [LHFC+16] identified, it is a productive research field, where main contributions have been focused on CIT approaches. When these approaches are used with large-scale real scenarios, scaling limitations have risen. Subsequently, approaches to solve these problems have been proposed [OMR10, JHF12], as well as alternative proposals [HPP+13b, HPP+13a] with other criteria to select products. However, as relevant works highlight [LHFRE15, dCMMCDA14], for the resolution of the product selection problem, CIT can be considered *de facto* as the standard technique.

Over the last years, research on **product prioritization** has been intensified, as it was described in Section 3.3. Two main approaches help performing the product suite sorting. Al-Hajjaji et al. [AHTL+16] proposed an approach to select and sort products based on the similarity criterion and before the products are generated. Sanchez et al. [SSRC14a] proposed five criteria to perform the sorting of already generated product suites. We can conclude in both approaches that product prioritization accelerates fault detection. The analyzed approaches that speed up fault detection are fundamental for the proposed approach of this research project.

Even after product selection and prioritization is performed the number of products to thoroughly test can result unfeasible for given time-budgets. Consequently, a need to propose new approaches and strategies to optimize the testing of products is emphasized.

3.6.2 Test case selection, minimization and prioritization

In contrast to the DE field, at the AE the research effort has not been so productive [LHFC+16]. Nevertheless inspiring improvements have been published in recent years.

In Sections 3.4 and 3.5 a summary of **test case selection, minimization and prioritization** is presented. Wang et al. [WAG15] provided a complete methodological and technical solution for selecting, minimizing and prioritizing test cases, as well as for developing and integrating the required tool chain to automatize the steps. Moreover, the approach was successfully evaluated with industrial studies [WAGL15, WAGL16b]. Arrieta et al. [AWSE16a] proposed test case selection and prioritization focusing in the CPS product lines for simulation-based validation contexts. The solutions results were also complete and have been validated with several case studies.

Nevertheless, when the HCCPS is large and testing time budget limited these approaches can result insufficient to ensure the required quality. Consequently, a need to propose new approaches and strategies to optimize the testing is highlighted.
Along with the identified need to improve the optimization of HCCPS testing we have also identified that the field corresponding to studying the optimization of the selection and prioritization of both test cases and products (together) has received little attention.

Chapter 4

Theoretical framework

Contents

This chapter gives a theoretical overview of the dissertation. To this end we define four research objectives (Section 4.1) together with the hypotheses (Section 4.2). Furthermore, we provide an overall explanation of the theoretical framework proposed for test optimization of highly-configurable cyber-physical systems (section 4.3). Finally, we describe the case studies employed to validate the effectiveness of the proposed solutions and the process conducted to design and build the cases studies (Section 4.4).

4.1 Research objectives

The main objective of this thesis is stated as follows:

> To provide a set of methods to maximize fault detection when testing highly-configurable cyber-physical systems in time-constrained scenarios.

This main objective is divided into the following operational objectives:

- *Objective 1:* Develop and evaluate test case **selection** algorithms for HCCPS.
- *Objective 2:* Develop and evaluate test case **prioritization** algorithms for HCCPS.
- *Objective 3:* Design and evaluate **iterative** approaches for efficient testing of HCCPS. Hence, design and evaluate strategies that enable efficient testing of HCCPS by chaining multiple testing iterations in which a small number of test cases are used each time.
- *Objective 4:* Design and evaluate **dynamic** approaches for efficient testing of HCCPS. Specifically, to design and evaluate efficient testing strategies that allow testing plans to be modified as they are being executed.

4.2 Research Hypotheses

Based on the objectives described in the previous section, the following hypotheses have been defined:

- *Hypothesis 1:* Combining domain-level information with application-level information using search-based techniques provides an iterative selection of test cases of a HCCPS that improves fault detection over traditional techniques. This hypothesis is related to research objectives 1 and 3.
- *Hypothesis 2:* Considering the structural coverage information of a HCCPS at Domain level helps optimizing fault detection results for time-budget constrained scenarios. This hypothesis is related to research objective 1.
- *Hypothesis 3:* Considering the results of tests executed on products, allows a dynamic prioritization of tests of a SPL that improves the capability to detect faults. This hypothesis is related to research objectives 2 and 4.

4.3 Theoretical Framework Overview

The optimization of HCCPSs testing presented in this work is based on developing methods to improve the SPL test optimization reference process adapted to the specific field of configurable CPS. As the proposed methods rely on a *core concept* that has inspired the research work presented in this thesis, this section is organized in such a way that we first introduce the core concept to later detail the theoretical framework.

4.3.1 Core Concept

In essence, the research work of this thesis has been developed on the basis of one *core concept: product and test cases combined test optimization.* This core concept is motivated by the difficulty to test HCCPSs exhaustively with the reference testing process (described in Section 3.1.1) and the need to provide alternative cost-effective testing methods.

Most of the SPL community's effort to optimize testing has focused on deriving a representative set of products of the product line which can be tested affordably [LHFC+16, dCMMCDA14]. Once this set of products has been obtained, the thorough testing of each individual product is typically performed. For instance, let us introduce the following abstract example.

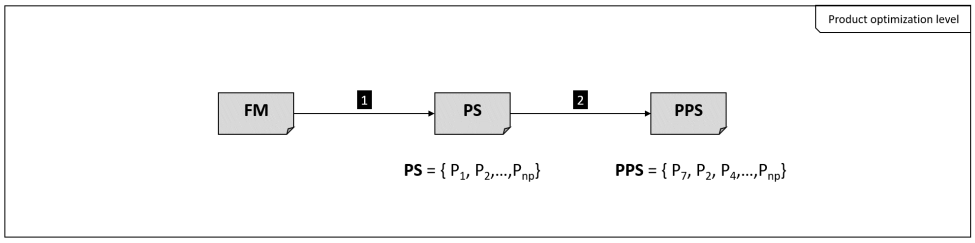


Figure 4.1: Product optimization steps.

Let FM be the *Feature Model* with which we represent all the products of a given product-line. And let $PS = \{P_1, P_2, ..., P_{np}\}$ be the *Product Suite* to be tested obtained from deriving the FM (step 1 in Figure 4.1), being np the total number of products. And let $PPS = \{P_7, P_2, P_4, ..., P_{np}\}$ be the *Prioritized Product Suite*, obtained from reordering the PS (step 2 in Figure 4.1), containing the same np number of products. Note that the sub-index of products in PPS have been reordered.

Let $TS = \{TC_1, TC_2, \cdots, TC_{nt}\}$ be a test suite with nt number of test cases generated to test products of the product line. Note that TS could be

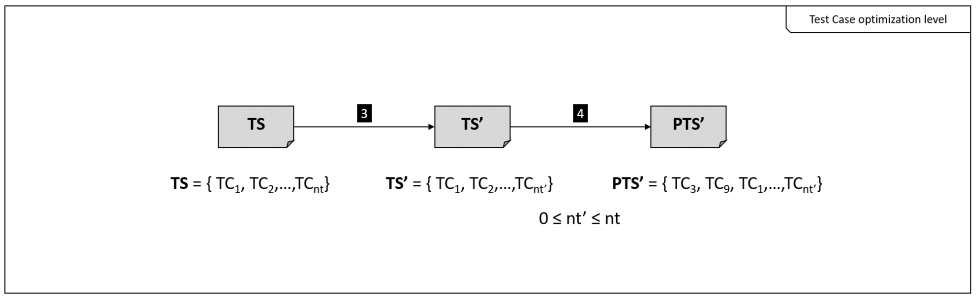


Figure 4.2: Test cases optimization steps.

generated to test any product of the whole PS or also could be generated to test a specific product of the PS). In addition, let $TS' = \{TC_1, TC_2, \cdots, TC_{nt'}\}$ be a subset of *test cases selected* to test the PS (step 3 in Figure 4.2), being nt' the number of test cases selected $0 \leq nt' \leq nt$. Finally, let $PTS' = \{TC_3, TC_9, TC_1, \cdots, TC_{nt'}\}$ be the new ordering of TS' test cases after applying the *test case prioritization* (step 4 in Figure 4.2).

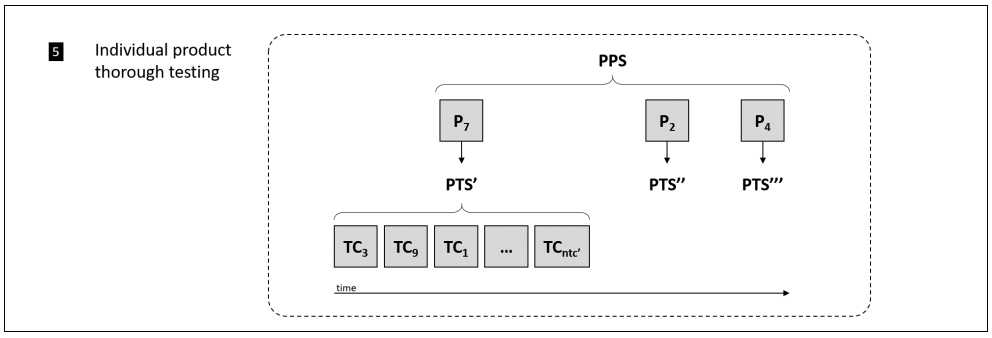


Figure 4.3: Product and test cases execution.

When executing products and related test cases (step 5 in Figure 4.3), the traditional testing approach selects products based on their prioritization (i.e., select products according to the order established in the PPS). For each product, the set of test cases to be applied is selected and prioritized (in Figure 4.3 PTS' is selected for product P_7), and all test cases are exhaustively executed. Once the first product is completed, the next prioritized product (P_2 in the example of Figure 4.3) is taken. This process will continue until all products are completely tested or available testing time-budget expires.

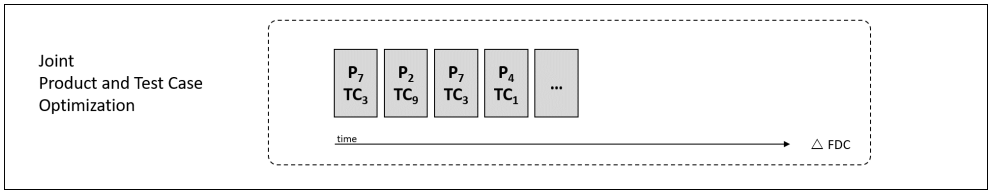


Figure 4.4: Joint product and test cases optimization.

The fundamental idea behind the thesis is to perform the selection and prioritization of products and test cases together. The approach proposed in this thesis selects and prioritizes products and test cases in a combined way, looking to optimize the ability to reveal faults in a given time. Following the example of this section, it is possible to select and prioritize a sequence of products and associated test cases ($P_7 - TC_3$, $P_2 - TC_9$, $P_7 - TC_3$, $P_4 - TC_1$, etc.) that as a whole are able to detect more faults for the given time than the traditional testing approach. The general idea, depicted in Figure 4.4, has been named *core concept* and is defined as follows:

> *The **core concept** proposes the possibility of **combining products and test cases** when selecting and prioritizing in a dynamic and iterative way. This selection and prioritization of test cases and products **considers product line information at both domain and Application levels** to enable the allocation of small groups of test cases and products in a cost-effective way.*

4.3.2 Theoretical Framework

The methods proposed in this thesis develop the *core concept* and have been structured based on the SPL test optimization reference process. Figure 4.5 presents these steps of the product line testing in a synthetic way in order to focus the work proposed in this thesis. Detailed explanation of the selected steps for the HCCPSs testing is provided in Section 3.1.1.

The proposed methods are grouped into two stages. In a first exploratory stage (blue colored in Figure 4.5), two methods were developed with the objective of exploring how to select products and test cases in a combined manner and how to combine information from the Domain level and the Application level to optimize the selection. The first exploratory contribution was addressed with black-box testing metrics, while the second, employed white-box testing metrics. After corroborating the first and second hypotheses during the experimentation of the exploratory stage, we designed and conducted the second stage, where the method for dynamically prioritizing tests was carried out as the main contribution of this thesis (green colored in Figure 4.5). The main contribution was employed to confirm the third hypothesis.

- The method entitled *Search-Based Test Allocation for Iterative testing of HCCPS* is part of the undertaken exploratory research (blue colored in figure 4.5). This method first performs product selection and prioritization by using well-proven techniques in the literature of product lines [JHF12, SSRC14a]. Then, we propose a novel approach to allocate small number of test cases in each product for iteratively testing the product line, thus replacing traditional steps for test case selection, prioritization and execution (i.e., steps number 3, 4 and 5 in figure 4.5). Further details of the method and obtained results are provided in chapter 5.

- The method named *Test Case Selection of HCCPS using Structural Coverage* is also part of the undertaken exploratory research (blue colored in figure 4.5). The method was developed to evaluate different test case selection methods based on structural coverage of both Domain and Application levels of the product line. The research performed in this method replaces the traditional test case selection step (i.e., step number 3 in figure 4.5) and modifies the traditional product prioritization result (i.e., step number 2 in figure 4.5). It is further detailed in chapter 6.

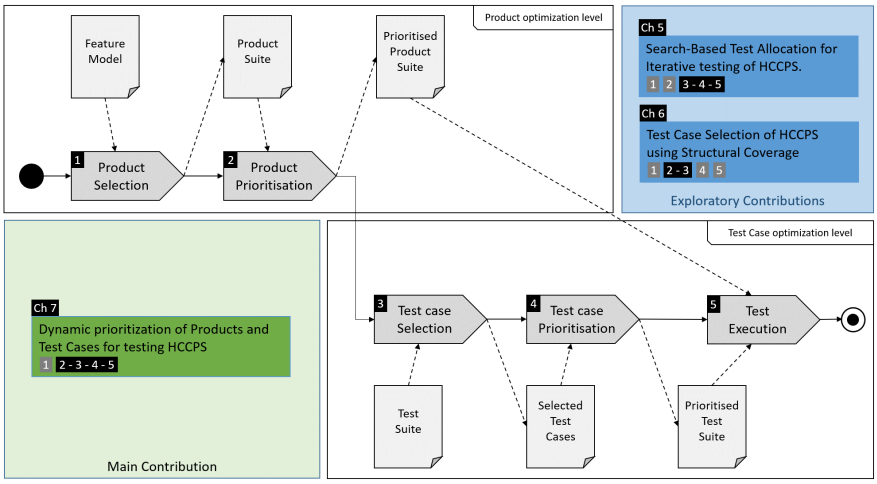


Figure 4.5: Traditional product line test optimization steps and the three methods researched. Traditional product line test optimization is represented in 5 steps (grey colored) that are further described in Section 3.1.1. The first exploratory method "Search-Based Test Allocation for Iterative testing of HCCPS" (blue colored) involves steps number 3, 4 and 5 and is further described in chapter 5. The second exploratory method "Test Case Selection of HCCPS using Structural Coverage" (blue colored) involves steps number 2 and 3 and is further described in chapter 6. Finally, the "Dynamic prioritization of Products and Test Cases for testing HCCPS" (green colored) method involves steps number 2, 3, 4 and 5 and is further described in chapter 6.

- The *Dynamic prioritization of Products and Test Cases for testing HCCPS* method is presented as main contribution of this thesis (green colored in figure 4.5). The method proposes a dynamic test prioritization (of both products and test cases) that leverages information of test cases being executed in specific products. The research performed in this method replaces traditional steps for test case selection, prioritization and execution (i.e., steps number 3, 4 and 5 in figure 4.5) and modifies the traditional product prioritization result (i.e., step number 2 in figure 4.5). Further details of the approach and results are provided in chapter 7.

4.4 Case Studies

This section presents the description of the case studies employed to validate the effectiveness of the proposed approaches, identifying which of the case studies was used with each of the contributions. Moreover, the semi-automated process conducted to elaborate the case studies is outlined.

4.4.1 Case study automation

Three case studies have been used in this thesis for evaluation purposes. Each case study consists of a HCCPS implemented with a configurable simulation model. Since the original case studies were lacking variability (i.e., they were not product-lines), it has been necessary to incorporate variability into the original case studies.

Based on the designed FM, the 150% Model of the product line has been developed.[1] From the 150% model, both the 150% model of the harness (i.e., test suite) and the 150% model of the mutants have been generated. The models of the specific products, the mutants that apply to them and the specific test cases for each products have been derived using the information of the product configurations. Figure 4.6 depicts the case study building steps. The remainder of the sub section details each step along associated implementation concepts.

- **Variability modeling**

  Feature Models are widely employed in the literature for variability representation (Refer to Section 2.2.2 for further details). *FeatureIDE* [TKB+14] is a remarkable tool integrated into Eclipse "ecosystem" that provides (among others functionalities) a powerful framework for modeling the FM and generating variants. Since the original models on which the case studies have been based were single (product) models, it has been necessary to capture their variability. To this end a FM diagram developed with *FeatureIDE* has been employed (Represented in step (1) of Figure 4.6).

  Let PL_{cps} be a product line of CPSs where each configuration is a specific CPS product. Let $FM = \{f_1, f_2, \cdots, f_{nf}\}$ be a feature model, with a set of features (F) and constraints, that represents the variabilities and commonalities of the PL_{cps} product line in a compact form, where nf is the number of features.

[1] The 150% model concept is described later in this sub section.

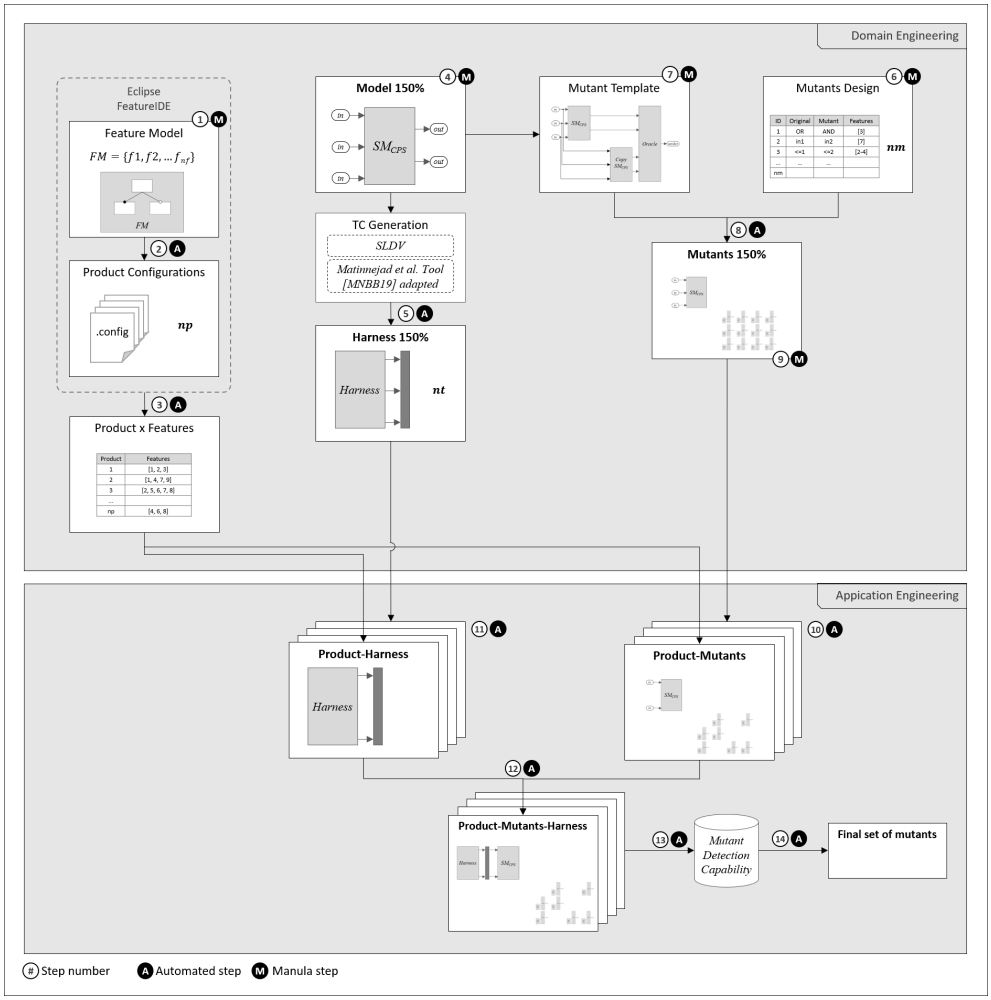


Figure 4.6: Case studies building automation steps.

■ **Product configuration**

The generation of product configurations, depicted in step (2) of Figure 4.6, has been performed in an automated way using the *FeatureIDE* Variant Configuration Generator wizard. This wizard is a flexible tool where the strategy to generate variants and the algorithm to apply for generation can be customized. The strategy can be set among others, to create all valid configurations, or cover feature interactions with a degree of T-wise (i.e., T=2 is set up to obtain pairwise feature interaction coverage). The tool also provides multiple research algorithms if T-wise strategy is selected (e.g., ICPL, Chavatal, CASA, IncLing). Refer to [MTS+17] for further details of the tool. When generating product configurations, pairwise coverage has been selected as it guarantees an appropriate level of feature interaction

coverage and generates a feasible number of product configurations for the development of the research. In addition, to satisfy pairwise coverage the ICPL[2] algorithm has been employed, which has proven to provide good results [JHF12]. As a result of the generation a *np* number of product configurations were created per case study.

Since the tool used to design and run the simulation models has been *Simulink* and the tool used for the automation of the different artifacts created for the product line has been *MATLAB*, the information of features and product configurations was externalized from *FeatureIDE* for its treatment in the *MATLAB/Simulink* environment. Data extraction is identified as step number (3) in Figure 4.6.

■ **Simulation modeling**

Let SM_{cps} be an abstract simulation model designed to address the behavior of an specific CPS, where $ISig = \{Isig_1, Isig_2, \cdots, Isig_k\}$ is the set of input signals and $OSig = \{Osig_1, Osig_2, \cdots, Osig_j\}$ the set of output signals. Input and output signals are vectors of time-indexed values. The number of vector values depends on the number of samples (i.e., time steps) observed during the simulation [MNBB16]. For a given fixed simulation time, the higher the number of observed sample values, the higher the precision of the simulation. For example an abstract simulation model is depicted in Figure 4.7 with a set of three input signals and a set of two output signals.

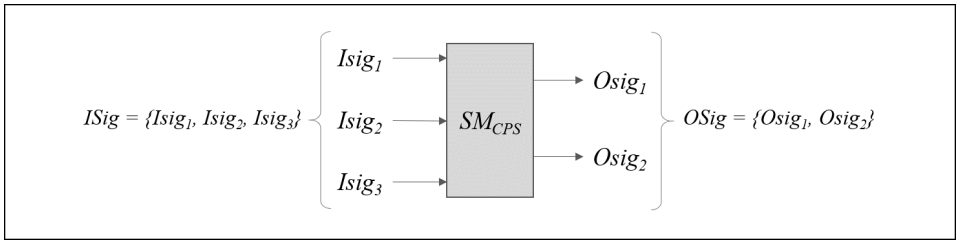


Figure 4.7: Simulation model of an abstract CPS example containing a set of three input signals and a set of two ouput signals.

Simulink is a tool set integrated into MATLAB tool set that provides model-based design tools for early testing [Mat21]. The tool graphically assists modeling of the system under test and the physical plant. It provides solvers for modeling and simulating dynamic systems. Code generation from models is supported for embedded processors. A complete set of packages with multiple purposes blocks and connectors are provided with customization

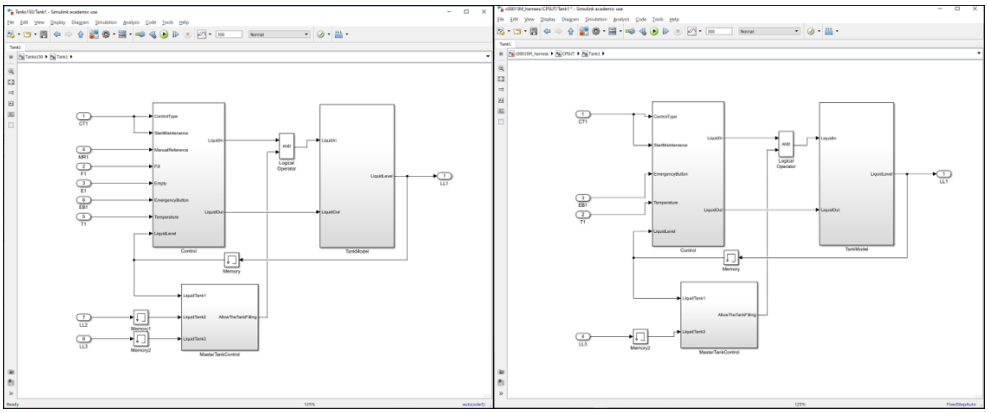
[2]ICPL is a recursive acronym that stands for "ICPL Covering array generation algorithm for Product Lines"

options. An Application Program Interface (API) is also provided to program and automate functionality.

MATLAB/Simulink has been employed for the experimentation carried out in this thesis, not only because of the facilities provided by its complete tool set but also because it is a tool that can be used both in the academic world for research and in the industry for manufacturing purposes. The explanations in this section will focus on the MATLAB/Simulink tool to support the description of the experimentation carried out in Chapters 5, 6 and 7. However, other alternatives could be used.

- **150% Model**

  150% models are artifacts that represent all the variability of the product line in a single domain model [LS14, AWSE16a]. When working with 150% models, features are related to specific blocks of the 150% model. Thus, to derive a particular product, negative variability can be applied [VG07]. Based on the product configuration, blocks of the 150% model are selectively eliminated and configured until the specific product (i.e., so called 100% model) is obtained. Figure 4.8 provides an example of the Industrial Tank case study. Left, the 150% Model (Figure 4.8a) is presented, while on the right a particular product model (Figure 4.8b) obtained from applying negative variability is shown.



(a) 150% Model **(b)** Specific product (100%) model

Figure 4.8: Example of simulation model product obtained from applying negative variability to the 150% model. Note that specific product model (Figure fig:negativevarapplied) contains less blocks and signals.

In this thesis we have employed a 150% model to represent all the variability of each HCCPS. Based on the FM, a 150% model has been manually produced for each case study and features related to specific blocks. The elaboration of the 150% model is depicted as step (4) in Figure 4.6.

Let $SM_{150\%}$ be the 150% simulation model that implements all possible configurations of the PL_{cps} product line into a unique model. Let $ISig_{150\%} = \{Isig_1, Isig_2, \cdots, Isig_{NIsig}\}$ be the set of input signals of the $SM_{150\%}$ model where $NIsig$ is the number of input signals.

■ **Testing simulation models**

Let T be the simulation time of SM_{cps} model. Let Δt be the fixed time step (i.e., fixed observed sample time length) where $T = \sum_1^k \Delta t$ and k is the number of samples observed during the simulation. A specific input signal $Isig_1$ of SM_{cps} model is composed of $k+1$ vector values $Isig_1 = \{V_0, V_1, V_2, \cdots, V_k\}$. Following the abstract example of Figure 4.7, a test harness containing one test case designed for the SM_{cps} sample model composed of three signals (input vectors) is presented in Figure 4.9, with a fixed time step of $1ms$.

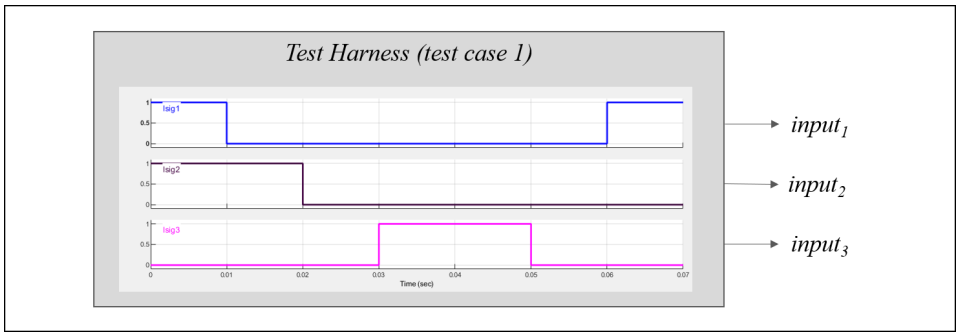


Figure 4.9: A harness block containing a test case. The test case is composed of three signals (input vectors). The fixed time step is set to $0.01sec$, thus, each input vector provides 8 values during simulation.

When testing a simulation model, the test harness containing test cases is linked to the simulation model, usually through a block to ensure data type casting and deterministic data transfer (dark grayed in Figure 4.10). When the system is simulated, designed test cases (signal inputs) stimulate the model and results are measured through the output signals of the model.

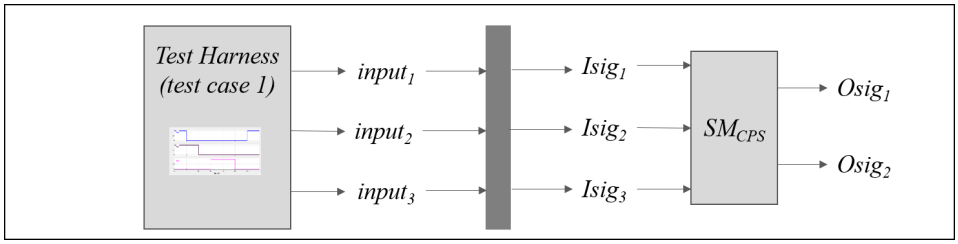


Figure 4.10: Simulation model CPS testing abstract example. A harness block containing one test case, linked to the data transfer subsystem, is also linked to SM_{cps} model input ports. To execute the test case, the simulation is run and the test case stimulates the model. Results are measured through output signals of the model.

- **Test case generation**

  The 150% model has been taken as the basis for test case generation. Taking into account the design of the 150% model, three different strategies have been used for the generation of the test cases. Firstly, the domain knowledge has been employed to generate test cases. Secondly, using the Simulink Design Verifier (SLDV) tool (i.e., a tool integrated in the MATLAB/Simulink tool set), the automatic generation of the test cases has been performed to satisfy the MC/DC type structural coverage (Refer to Appendix A for further details of the MC/DC structural coverage criterion). Finally, the MATLAB/Simulink tool proposed by Matinnejad et al. [MNBB19] has been adapted for test generation in order to randomly generate valid test cases for the 150% model. The test case generation is depicted as step (5) in Figure 4.6. Note that the SLDV and *Domain Knowledge* have been used for the test case generation during the exploratory contributions (described in chapters 5 and 6), whereas the tool developed for the generation of valid random test cases for 150% models has been used for the main contribution (described in 7). As a result, one 150% model of the harness was obtained per case study.

  Let $TS_{150\%} = \{TC_1, TC_2, \cdots, TC_{nt}\}$ be the (harness) test suite containing a nt number of test cases generated for testing the $SM_{150\%}$ of the product line.

- **Mutant design**

  Since no real faults were available for the case studies, mutation testing was applied, which has proven to be a good substitute [JJI+14]. As described in Section 2.4.2, this technique generates a copy (*mutant*) of the system in which a fault (*mutation*) is injected. If, when running a test case, a certain output value of the copy differs from the original system, the mutant is considered to have been *killed*, hence the fault detected. As case studies of this thesis are based on HCCPSs implemented as configurable simulation models, when designing mutant injection three criteria were defined: (i) distribute faults on the 150% model, (ii) inject faults in different types of blocks and (iii) employ suitable mutation operators for simulation models. Specifically, we employed mutation operators proposed by Hanh et al. for Simulink models [HBT16]. Taking into account these criteria, a *nm* number of mutants was designed for each case study. Additionally, each fault was related to a specific feature associated to the 150% model. The design of mutants to be injected is shown in step (6) of Figure 4.6.

- **Mutant template generation**

  In order to automate the generation of (i) the mutant systems (i.e., the copies of the original systems) and (ii) the oracles (i.e., the copies of the mechanism for checking the outputs of the systems and determining the verdict), a mutation template was developed manually for each case study.

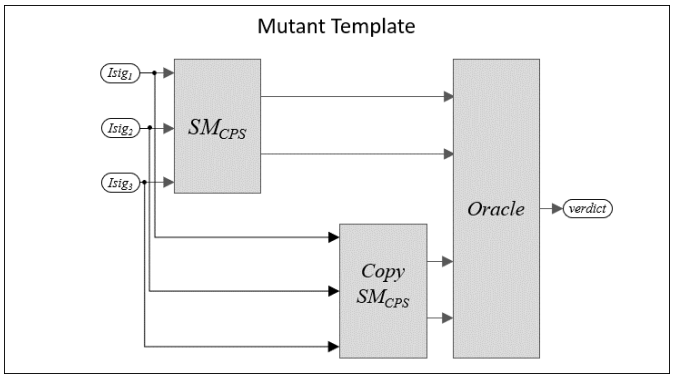


Figure 4.11: Abstract mutant template example containing 5 key elements: (i) an original simulation model SM_{CPS}, (ii) a mutant $Copy\ SM_{CPS}$ simulation model, (iii) a set of input signals $Isig_{SM_{CPS}}$ shared by both original and mutant models, (iv) an *Oracle* for the evaluation, and (v) a *verdic* block to store oracle result.

Note in the abstracted example of Figure 4.11 that both the original model (SM_{CPS}) and the mutant ($Copy\ SM_{CPS}$) share the same input signals (i.e.,

$Isig_{SM_{CPS}} = \{Isig_1, Isig_2, Isig_3\}$). When a test case is executed, the results of the original model and the mutant are processed by the *Oracle* block and the *verdict* is stored. It is important to note that this mutant template does not contain faults. This template is an artifact to (later) dynamically generate the models that will contain the faults and the corresponding oracles. The injection of the faults is described in later steps. The development of the template is shown in step (7) of Figure 4.6.

- **Mutant 150% model generation and fault injection**

  A tool was developed to generate the 150% mutant model. As input to the tool, the mutant template and the desired number of mutants are provided. The tool generates a 150% mutant model, which contains the original system, as well as a number of copies of the mutated system with their corresponding oracles. The abstracted example of Figure 4.12 shows the original system SM_{CPS} and a nm number of copies of the system with their respective oracles (i.e., 12 copies of the mutant and oracle are presented in the figure). Required shared input signal connection to all mutants representation is avoided in the figure for the sake of clarity. However, a representation similar to the one made in the mutant template can be intuitively guessed. The generation of the 150% mutants model is indicated as step (8) of Figure 4.6.

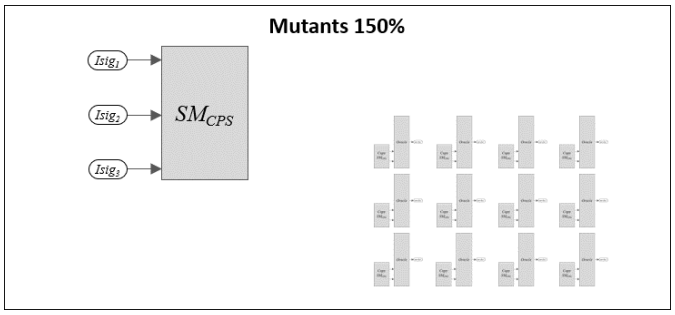


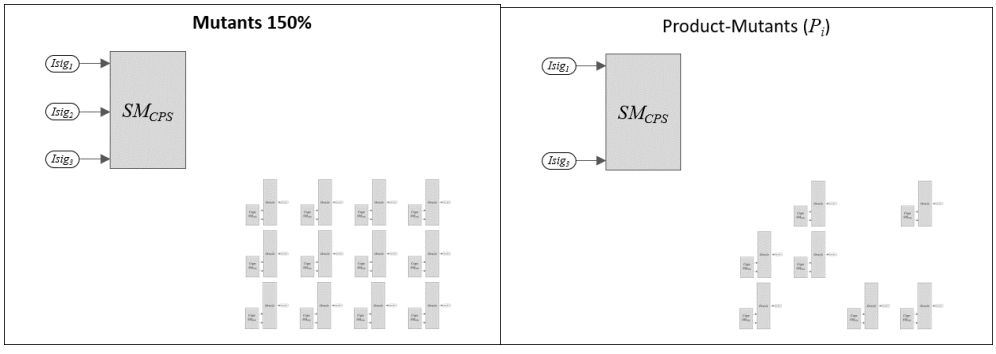
Figure 4.12: Abstract representation of the Mutants 150% model. The example provides the original simulation model SM_{CPS} and a nm number of mutants (i.e., copies of the system) with their respective oracles. Shared input signal connections are not represented for the sake of clarity.

Unlike purely software-based environments, we lack tools for automatic mutant injection in CPSs based on simulation models, thus making mutant injection costly. Therefore, manual injection turned out to be more feasible than developing a tool to automate it. On the basis of the designed mutants, a nm number of mutants were injected into each mutant 150% model. The mutant injection is depicted as step (9) of Figure 4.6.

■ **Product and Mutants derivation**

To derive the specific *np* products to be generated in the product line, a tool was developed. The tool takes as input the Mutants 150% model and the product configurations information from step 3. The Mutants 150% model includes the Model 150% of the product line and a *nm* number of mutants (as described in step 9). The tool applies negative variability to the Mutants 150% model and, taking into account features of the product to be derived, keeps those blocks that are related to the selected feature, whereas the blocks related to non-selected features are removed. With regard to the mutants, only those mutants related to features included in the product configuration are maintained. Moreover, in the same way the negative variability was applied to the product, each mutant model is also adapted, by removing unnecessary blocks. As a result, *np* number of models are created. Each of the created models contains a specific product model and a number of mutants. Step (10) of Figure 4.6 depicts product derivation.

Let $SMPS = \{P_1, P_2, ..., P_{np}\}$ be the product suite of simulation models to be tested, derived from the $SM_{150\%}$ product line (i.e., Model 150%) following any state-of-the-art product derivation approach [CDS08, POS+12, HPP+14], where *np* is the total number of resulting products. Let $ISig_{P_i} = \{Isig_1, Isig_2, \cdots, Isig_{NIsig_{P_i}}\}$ be the set of input signals of P_i product, where $0 < NIsig_{P_i} \leqslant NIsig$. Let $SMMS_{P_i} = \{M_1, M_2, \ldots, P_{nm'}\}$ be the mutants suite of simulation models derived along with the product P_i, where nm' is the number of mutants obtained for the specific product being $0 \leq nm' \leq nm$.



(a) Model 150% with all mutants. **(b)** P_i product with related mutants

Figure 4.13: Example of applying negative variability to the Mutants 150% model. Right figure shows derived product P_i and related mutants.

For example, Figure 4.13 presents an abstract derivation exercise. The figure on the left (Figure 4.13a) shows the Mutants 150% model containing the product line and all possible mutants, whereas the figure on the right (Figure 4.13b) the model obtained from deriving product P_i is presented. Note that in the figure on the right, certain input signals and mutants are missing, a consequence of applying negative variability.

- **Test suite derivation**

  We developed a tool to derive the specific test suites for the generated products. Said tool takes as input the Harness 150% model and the product configurations information from step 3. Taking into account the features of each product configuration, the tool generates a specific harness with nt number of test cases adapted to the signals of the product model. Step (11) of Figure 4.6 depicts test suite derivation. Note that for the main contribution, all test cases designed in the Harness 150% were valid for all products. However, it is common that not all test cases in the test suite are valid for all products. For example, in the exploratory-stage contributions, each product had a number of test cases (nt') less than or equal to the number of test cases of the Harness 150%.

  Let $TS_{P_i} = \{TC'_1, TC'_2, \cdots, TC'_n t\}$ be the particular test suite composed of nt test cases adapted from $TS_{150\%}$ to test the P_i product. Note that a test case generated for the $SM_{150\%}$ model (e.g., TC_1) consists of $NIsig$ input signals and is instantiated to a new test case (e.g., TC'_1) with $NIsig_{P_i}$ number of input signals of the particular product P_i.

- **Model integration and test execution**

  A tool was developed to integrate the product, related mutants and the harness into one model per configuration. Thus, harness signals (adapted to each product) got connected simultaneously to the product and to each of the mutants. Integration is depicted in Step (12) of Figure 4.6.

  Integrated models were next executed, i.e., the test cases contained in harness were executed with respective product models and mutant models and results were analyzed to gather mutant detection capability of each test case. Execution is depicted as Step (13) of Figure 4.6.

- **Mutant selection**
  To obtain the final set of mutants for the evaluation, three mutant selection criteria were applied: firstly, undetectable mutants were removed (i.e., mutants that were not detected by any test case), in order to avoid the inclusion of equivalent mutants. Secondly, duplicated mutants were removed (i.e., mutants equivalent to one another but not to the original model), as recommended by Papadakis et al [PKZ+19]. Lastly, we removed mutants that were easily detected by test cases. On average, remaining mutants accounted for 15% of the total initially designed mutants. Mutant selection is shown as Step (14) of Figure 4.6.

4.4.2 Description of case studies

Three case studies were employed for the experimental validation of the proposed contributions. Each case study is related to a different problem in the aerospace, automotive and industrial domain respectively. The complexity of the selected case studies is representatively significant for the chosen domains. All case studies are based on models published by the community, and adaptations were then implemented mainly to incorporate variability.

Unmanned Aerial Vehicle (UAV)

The UAV case study consists of a product family for a configurable quad-copter CPS. The original design was developed by Mosterman et al., who modeled a commercial UAV (i.e., the AR.Drone 2.0 from Parrot) with Simulink and integrated it into their fleet of autonomous vehicles for humanitarian emergency responses [MSB+14]. Mosterman et al. modeled the structure of the vehicle and the dynamics, in addition they empirically validated the mathematical equations for motion. Arrieta et al. [ASEZ16] extended the original model with variability points in both software and hardware to transform the *single* UAV model into a product line and employed it in several evaluations [ASEZ16, AWM+17b, AWSE16a]. The extended version modeled by Arrieta et al. was employed in this thesis.

- **System architecture** The case study is a configurable CPS composed by the physical layer and three platforms at the cyber layer. The overall architecture of the system is shown in Figure 4.14. Platform 1 performs the high level control of the system in order to make decisions about vehicle trajectory and communicating with the ground station. It is equipped with

an embedded computer for control purposes, three sensors to support the trajectory definition (obstacle sensor, battery sensor and GPS sensor), the communications module (to send information to the ground station) and a flight indicator LED. Platform 2 performs low-level control to ensure flight stability. From the system dynamics obtained with the gyroscope sensor data, it regulates the speed of each rotor to keep the UAV stable. The speed control of the rotors is performed on platform 3. It consists of the embedded computer and the rotors. Based on the instructions provided by the low level control, Platform 3 provides the speed commands to the rotors to maintain the flight of the UAV. The physical layer of the UAV consists of different physical processes, i.e., the dynamics, the global position and the battery level.

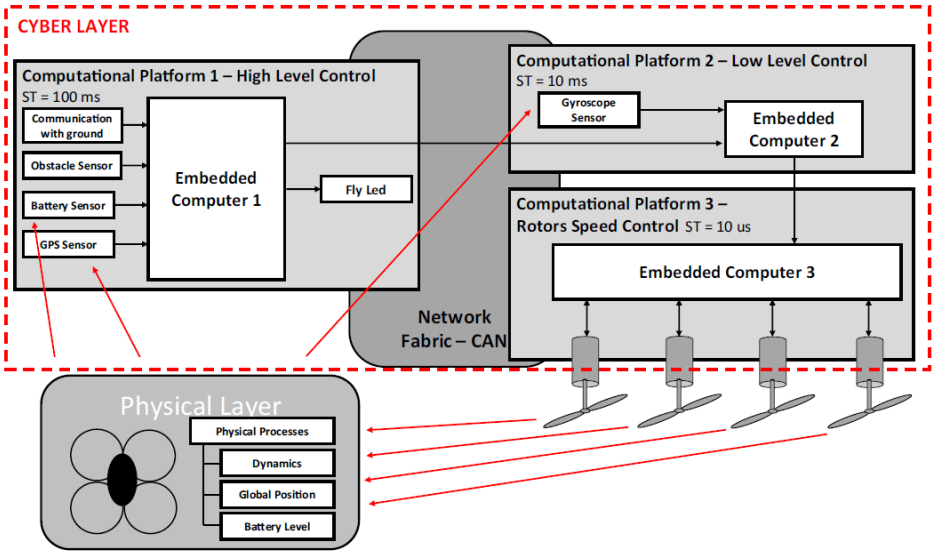


Figure 4.14: System architecture of the UAV case study. The architecture is organized in the physical and cyber layers. The cyber layer is composed of three platforms and the network fabric.

- **Variability** The variability of the case study is described in the feature model of Figure 4.15. The key variability aspects of the UAV configurable CPS are described as follows.

  - **Variability in Track System**: Three are up to three different track systems for trajectory planning that can be selected individually or in combination: (i) point-to-point, (ii) concrete coordinates and (iii) person following.

- **Variability in control strategy**: Two alternative options are provided to control position, height, speed and angles: (i) the proportional option and (ii) the proportional-integral option.
- **Variability in Safety functions**: There are up to four options to increase the safety functions of the CPS: (i) collision avoidance equipment, (ii) wind avoidance algorithm, (iii) an emergency system and (iv) back to home mode.
- **Variability in Battery Management**: Two alternative battery management modes are provided: (i) the auto-battery charge mode to find the closest battery station to recharge battery and (ii) battery landing program to ensure a sufficient time window to land the drone.
- **Variability in Operating System**: Two alternative options are provided in order to choose the operating system: (i) the manufacturer provided default embedded Linux and (ii) the FreeRTOS alternative.[3]
- **Variability in Battery Models**: Two alternative options are provided for batteries: (i) a *short* duration battery (approximately 12 minutes) and (ii) the *long* duration battery (approximately 25 minutes).
- **Variability in Sensors**: Up to four types of sensors can be configured in the UAV. Gyroscope and GPS are mandatory sensors, however battery and obstacle sensors are optional. For each sensor category three alternative models are provided.
- **Variability in Rotors**: Two different types of rotors can be selected: (i) the high speed rotors and (ii) the low speed rotors.
- **Variability in Signaling**: The LED light that indicates that the quadcopter is flying can be optionally selected.

[3]FreeRTOS is a cross-platform real-time operating system kernel for micro controllers. https://www.freertos.org/

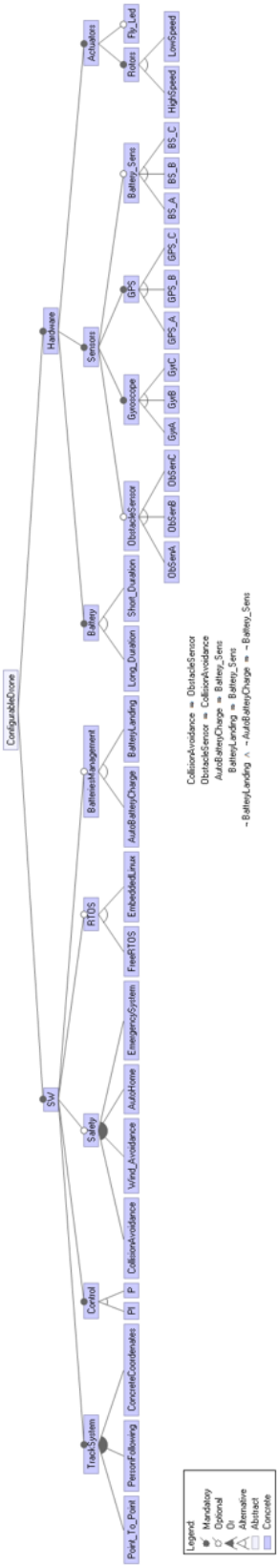


Figure 4.15: Feature Model of the UAV case study developed by Arrieta et al. [ASEZ16].

Car Windows (CW)

The CW case study consists of a configurable CPS product line for the control of car windows. This case study belongs to the automotive domain, where companies usually work with model-based cases. The original design was employed by Arrieta et al. [AWA+18] for single product evaluation purposes. We extended that one for this case study by including additional variability and transforming the system into a product line.

- **System architecture** The case study is a configurable CPS composed of the physical layer and two platforms at the cyber layer connected through CAN communications. The overall architecture of the system is shown in Figure 4.16. The physical layer involves the electrical and mechanical models of the four windows. On the cyber layer, Platform 1 performs the control of the system in order to decide whether to open or to close windows. It is equipped with an embedded computer for control purposes, two types of sensors and the interfaces with passengers. The embedded computer makes decisions regarding each window and hosts the prioritization algorithm. Obstacle sensors are provided for each window to detect the presence of obstacles that require interrupting the sliding of the window. Position sensors guide the opening and closing operations of the windows. The driver and passengers have different interfaces for specifying instructions (i.e., local window control, control of other passengers' windows or centralized controls). Decisions from Platform 1 are sent to Platform 2, which controls the operation of each window engine.

- **Variability** The variability of the case study is described in the feature model of Figure 4.17. Additionally, constraints to ensure the correct construction and functionality of the system are included. For instance, when child window type is selected, the control of the related window is mandatorily included in driver's control. The key variability aspects of the CW configurable CPS are described as follows.

  - **Variability in number of windows**: There are up to four windows and it is mandatory to include at least the two front seat windows (i.e., rear seat windows are optional).
  - **Variability in window controls**: There is a control system installed with each front-seat window to control the *local* window. Optionally, from one to all other windows can also be controlled, depending on the seat

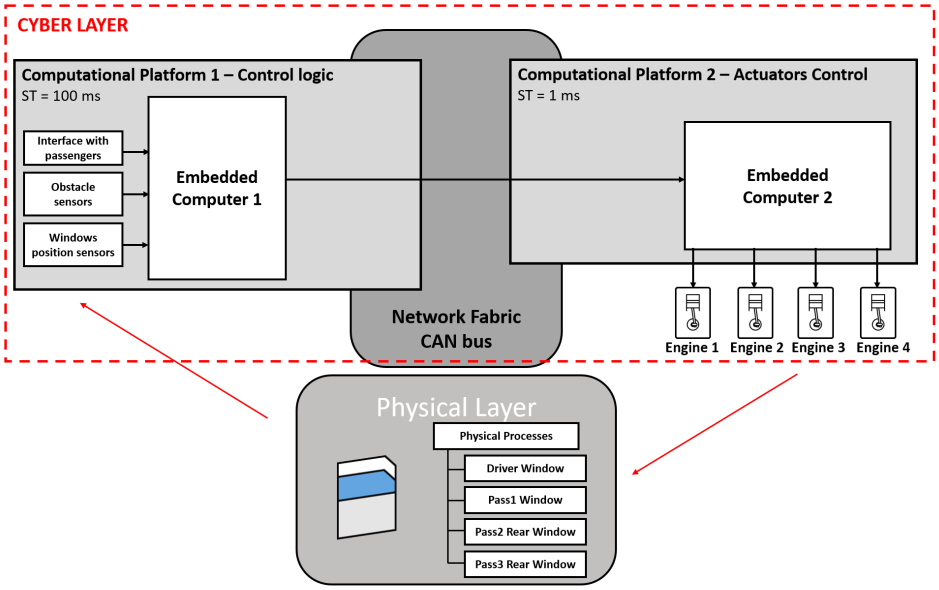


Figure 4.16: System architecture of the CW case study. The architecture is organized in the physical and cyber layers. The cyber layer is composed of two platforms and the network fabric.

type. For driver control, all other windows can be optionally selected to be controlled. For the co-driver control (identified as passenger 1), the rear seat windows can be optionally controlled.

- **Variability in rear-seat types**: There are up to three alternative options for rear seat windows: (i) the standard windows for passengers, (ii) the adapted safe window for children passengers and (iii) the commercial adaptation of the window for sliding doors. Specifically, while window3 can opt to any of the three alternative options, window4 can opt between the standard and children alternatives.
- **Variability in centralized control types**: There are two alternatives for centralized control of windows: (i) the option that integrates the centralized control of windows into the CentralBoard panel of the car and (ii) the remote centralized control of the windows' options. Additionally, each of the four windows to be controlled in a centralized manner can be optionally selected.
- **Variability in Lock**: There are three options that can alternatively selected for locking the windows: (i) Rear, to lock only rear windows, (ii) Child, to lock child type windows and (iii) Complete, to lock all available windows.

Industrial Tanks (IT)

The case study of Industrial Tanks was initially developed by Arrieta et al. in [ASE15] in the context of testing CPSs. It imitates the typical characteristics of an industrial tank CPS. This case study was adapted in order to experiment with the main contribution of this thesis. Thus, the variability of the system was increased, adapting the case study of a single product system to a product line. Specifically, it now deals with a three-tank product family instead of the original single-tank system. Additionally, control modes and prioritization capabilities were incorporated.

- **System architecture** The original case study was conceived as a CPS to meet industrial control requirements for individual fluid tanks. The cyber layer of the original model was composed of two independent platforms connected through EhterCAT communications. The first platform was dedicated to the processing of signals from the system's sensors. The data processed by this platform were then transmitted to the second platform, controlling the actuators to fill or drain gates. The original case study was extended to meet new requirements (mainly related to priority management subsystem). The resulting system architecture is depicted in Figure 4.18.

- **Variability** The feature model shown in Figure 4.19 describes the variability of the multi-tank configurable CPS and associated constraints. In addition to the construction restrictions of the feature model, functional restrictions are included, e.g. if the selected liquid is chemical, the temperature sensor must be included. The key variability aspects of the multi-tank configurable CPS are described below:

  - **Variability in number of tanks**: There are up to three tanks and it is mandatory to include at least one tank.
  - **Variability in control modes**: Each tank can have up to three control modes of which at least one must be selected. The operation control modes are: (i) Manual mode to start or stop filling or emptying the tank, (ii) Reference mode to establish a manual reference level of the tank to be filled and (iii) Maintenance mode, which runs a maintenance cycle empty-fill-empty.
  - **Variability in tank priorities**: There is an optional feature to prioritize a certain tank, i.e., the prioritized tank will always be the fullest one.

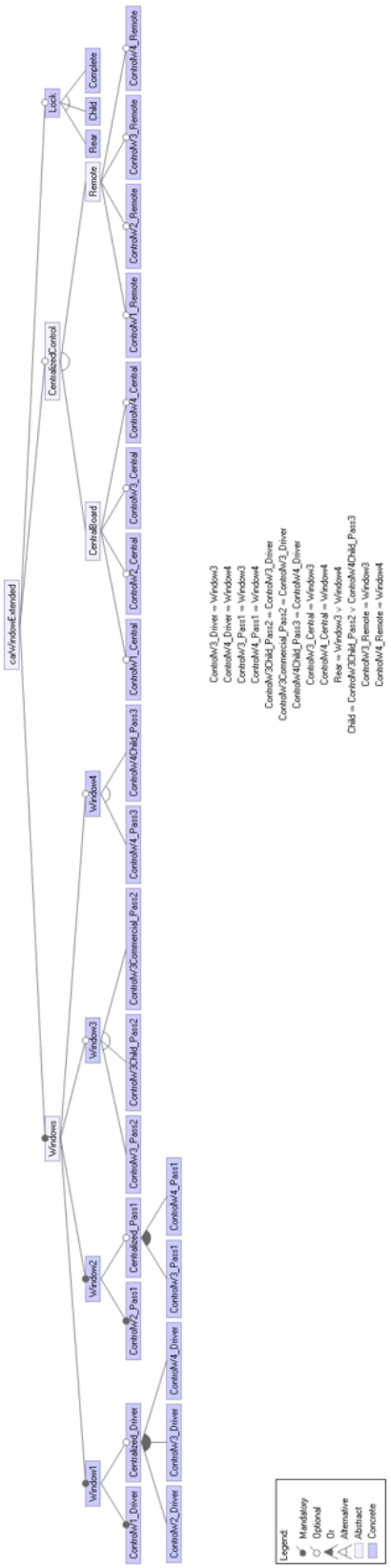


Figure 4.17: Feature Model of the CW case study.

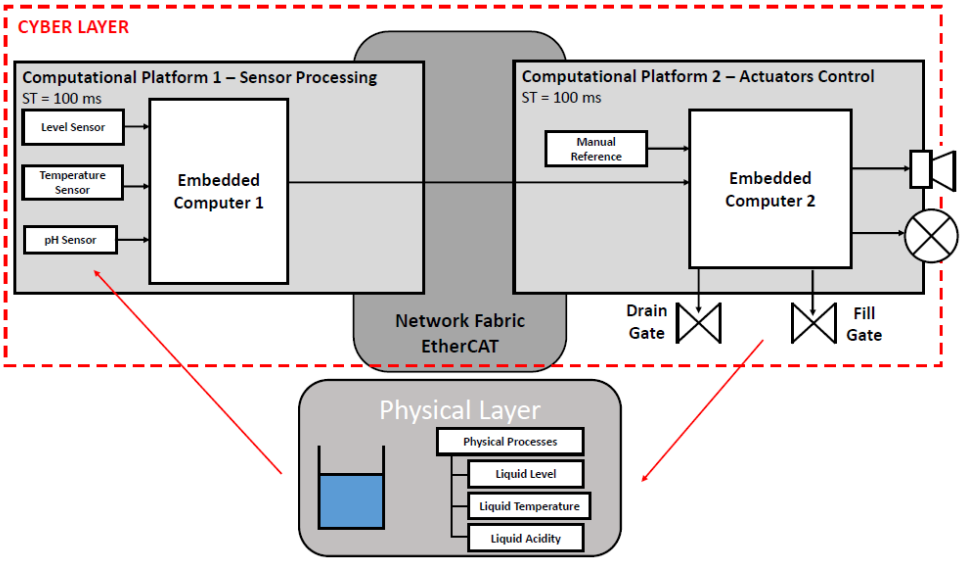


Figure 4.18: System architecture of the IT case study developed. The architecture is organized in the physical and cyber layers. The cyber layer is composed of two platforms and the network fabric.

- **Variability in sensors**: Each tank can have up to two sensors. The first sensor is mandatory and measures the liquid level of the tank. The second sensor is optional and can be employed to measure the temperature of the liquid.
- **Variability in liquid content types**: There are two liquid content types: water or chemical.
- **Variability in liquid flow speeds**: There are 3 liquid speeds, i.e., low, medium and high speed, when filling or emptying the tanks.

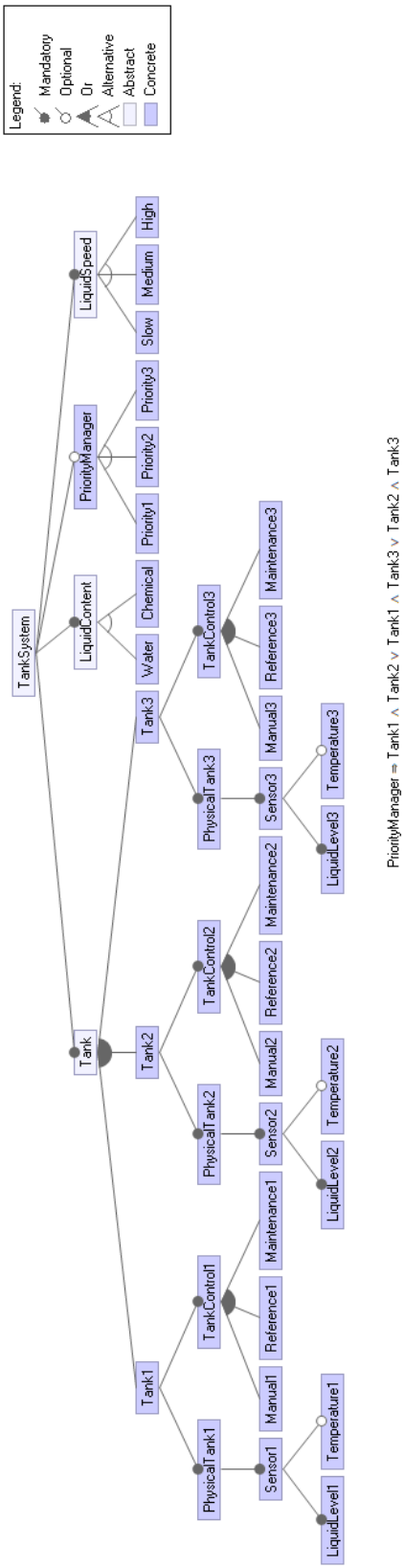


Figure 4.19: Feature Model of the IT case study.

Key characteristics of case studies

The selected case studies belong to various domains and have different characteristics in order to validate the approaches in diverse contexts. Tables 4.1 and 4.2 summarize the main characteristics of each case study. In Table 4.1 key characteristics of the simulation model, feature model and derived products are presented. Column `Blocks` indicates the number of blocks of each Simulink model, specifically the number of blocks of the 150% simulation model. Columns `Features` and `Constraints` refer to the number of features and constraints for each case study. With regard to the `Derived` column, it indicates the number of product configurations obtained from the FM applying pairwise technique, whereas the `All-Valid` column refers to the number of valid configurations that might be obtained from the product line.

Table 4.1: Key characteristics of the selected case studies with regard to the simulation model, feature model and derived products. The `Blocks` column provides the number of Simulink blocks for each of the case studies. The `Features` and `Constraints` columns specify the number of features and constraints for the corresponding feature model for each case study. `Derived` indicates the number of obtained products when deriving the product line, whereas `All-valid` refers to the number of valid configurations that might be obtained from the product line.

Case Study	Model150%	Feature Model		Products	
	Blocks	Features	Constraints	Derived	All-valid
Unmaned Aerial Vehicle (UAV)	843	46	5	22	800.000+
Industrial Tanks (IT)	306	36	7	17	11.824
Car Windows (CW)	227	30	13	28	39.582

Table 4.2 shows three columns corresponding to the generation of `Test Cases`. The `Gen.Criterion` column describes whether test cases were generated based on domain knowledge, based on the MC/DC structural coverage criterion or based on the tool developed to generate random test cases. The `150%` column indicates the number of test cases generated for the 150% model. If N/A is indicated, it means that the test cases were not generated for the 150% model (but directly for the derived products). The `Total` column indicates the total number of test cases generated after adapting 150% test cases to each derived product. The last two columns present data related to the `Number of Mutants`. The `Initial` column indicates the number of mutants initially designed and seeded for each case study and the `Final` column indicates the number of mutants remaining after the mutant selection process has been performed.

Table 4.2: Key characteristics of the selected case studies with regard to the generated test cases and mutants. Three columns describe the `Criterion` employed to generate test cases, the number of test cases generated for the `150%` model and the `Total` number of test cases after adapting 150% test cases to each derived product. The last two columns show the number of mutants per case study; the first column shows the number of `Initial` mutants generated, while the second column refers to the `Final` number of mutants employed in the evaluation after applying mutant selection.

Case Study	Test Cases			Number of Mutants	
	Gen.Criterion	150%	Total	Initial	Final
Unmaned Aerial Vehicle	Domain knowledge	N/A	120	N/A	20
Industrial Tanks	Random	150	2550	246	30
Car Windows	MC/DC	N/A	1184	200	129
Car Windows	Random	150	4200	209	36

Case studies employed in each contribution

To asses each contribution, an empirical evaluation assessment was conducted respectively.

In the first exploratory contribution *Search-based fault detection allocating small groups of test cases to products iteratively* (described in chapter 5), the UAV case study was employed. In the second exploratory contribution *Test case selection methods based on structural coverage of both the products and the product-line* (described in chapter 6), the CW case study was employed. In the UAV case study the *Domain knowledge* was employed for test case generation, while for the CW case study (employed in second exploratory contribution), the test cases were generated using the SLDV tool.

For the main contribution *Dynamic test prioritization approach of both products and test cases* (described in chapter 7), the CW and IT case studies were employed. With respect to test cases generation, the adapted tool for random valid test cases was employed.

Part II

HCCPS Test Optimization

Chapter 5

Search-Based Test Allocation for Iterative testing of HCCPS

Contents

This chapter provides details of the first exploratory research work. Section 5.1 contextualizes the presented contribution in the theoretical framework. Section 5.2 provides the technical motivation for this contribution. Section 5.3 presents our approach to optimize fault detection by allocating test cases for products iteratively. Section 5.4 exposes the experiment design, obtained results and overall discussion. Related work is reported in Section 5.5. Finally, conclusions are summarized in Section 5.6.

5.1 Contribution overview

The research work presented in this chapter details the first exploratory work of the thesis. Figure 5.1 highlights this work (in blue) in the context of the entire theoretical framework (further details of the theoretical framework are provided in Section 4.3).

This exploratory work was set in order to validate the first hypothesis of the thesis, which aims at improving the test case selection and prioritization of HCCPSs through an iterative approach employing search-based techniques.

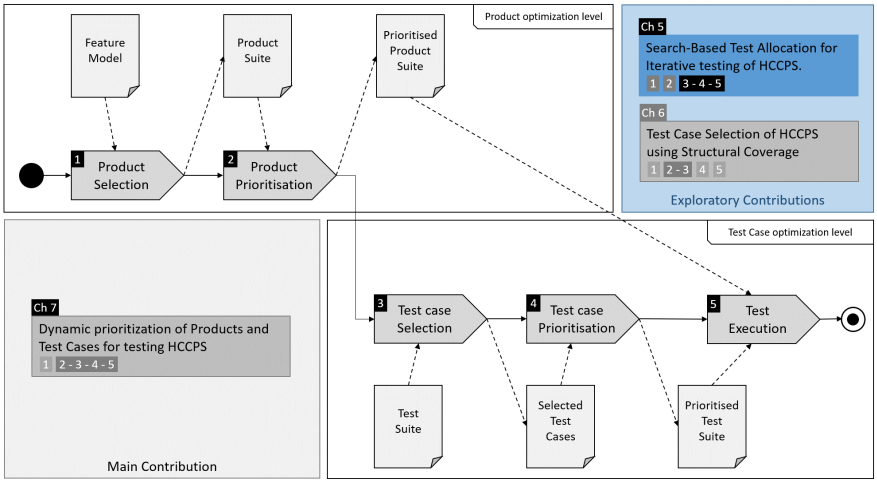


Figure 5.1: First exploratory method proposed in the context of the theoretical framework. The first exploratory method "Search-Based Test Allocation for Iterative testing of HCCPS" (blue colored), which involves steps number 3, 4 and 5.

The approach proposed in this exploration work is designed to develop the *core concept* (defined at the theoretical framework in Section 4.3) by combining the steps of selection and prioritization of test cases in an iterative way. Initially, the traditional techniques for the product selection and prioritization steps are

employed (i.e., well-known state-of-the art techniques are employed for steps 1 and 2 of Figure 5.1). Next, the proposed approach employs a novel search-based technique to optimize the allocation of small groups of test cases into pre-selected and pre-prioritized products (i.e., the search exercise merges steps 3 and 4 of Figure 5.1). The search and test (execution) process is iteratively repeated until the time budget is consumed.

We performed an evaluation with the UAV case study which is further detailed in Section 4.4. Results suggest that our approach can reduce the fault detection time on average by 55%; and by 76% when compared with the traditional test process and the Random Search algorithm respectively.

This research work allowed us to empirically validate the exploratory line of work proposed to optimize the testing of HCCPSs by allocating a **small number of test cases** for each product in an **iterative manner**.

5.2 Introduction

Testing product lines is a challenging process due to the high number of potential products that the system can be set to. As a result, recent research activities have focused on selecting relevant products to be tested from the product line, ensuring a certain degree of test coverage [PSK+10, LHLE15, AHTL+16]. Typically these approaches take a feature model as an input and, by applying Combinatorial Interaction Testing (CIT) algorithms, a set of related products satisfying all the constraints are obtained [PSK+10, CDS08, LHLE15]. The pairwise product generation criterion is usually employed since its use ensures that the interaction of two features will be covered at least once. This increases the chances of finding faults without the need for testing a large amount of products due to the interaction of two features. However, despite using this product derivation criterion, testing all these products is still challenging and thus, other techniques such as the test case selection [AWSE16a], minimization [WAG13, WAG15] and prioritization [WBA+14, AWSE16b] have been proposed.

Testing product lines typically follows a four-step process. First, when considering a variability model (e.g., a feature model), relevant products to test are derived based on some criteria (e.g., the smallest amount of products while ensuring pairwise coverage) [PSK+10, CDS08, HLL+16]. Secondly, these products are prioritized with the aim of increasing the fault detection rate [SSRC14a, SSPRC15, PSS+16, AHTL+16]. After prioritizing these products,

relevant test cases that must be executed in each of the products are selected at the application engineering level [SMP10, WAGL16b, WAGL15, AWSE16a]. Once these test cases are selected, the last step consists of their prioritization, where the order in which the tests must be executed to test the product is determined [WBA+14, AWSE16b].

Current product line engineering testing approaches focus on selecting relevant products to test and, following this, testing each of the products thoroughly. However, testing each of the individual products is not always trivial, since the test cases are not always fast to execute. In systems such as CPSs, where the testing is performed at system level, the test execution time for each test case can last thousands of seconds. Thus, even if test selection methods have been proposed to optimize the test execution time of individual products while ensuring high test quality [AWSE16a], this may not be enough for the product line engineering context. Specifically, even if some products could be thoroughly tested (probably the products that had a higher priority assigned), there might be some products (those scheduled to be tested last) that cannot be tested as the time budget may have expired by the time of testing them.

To solve this problem we propose a novel test methodology to iteratively test product lines. Specifically, we propose a search-based testing method that, given a product suite as an input, iteratively allocates a small number of test cases to each of them. Once the selected products are tested with the allocated test cases, the developed search algorithm is re-run to allocate other test cases in the same products. This way, instead of thoroughly testing each of the selected products individually, the test is performed in an iterative manner. This allows for the execution of at least some test cases in each of the selected products, reducing the risk of not testing some of the products due to constraints related to the test execution budget. We evaluated different types and configurations of Genetic Algorithms (GA) in an empirical evaluation with a CPS product line as a case study. The results suggest that our approach can help to improve the fault detection rate as compared with the traditional testing method as well as with Random Search (RS).

5.3 Search-based Test Allocation

The essential idea of the approach is to perform the testing of product lines by allocating a small number of test cases in each of the products. This approach increases the probability of detecting faults faster. The proposed approach follows five steps and an overview is depicted in Figure 5.2.

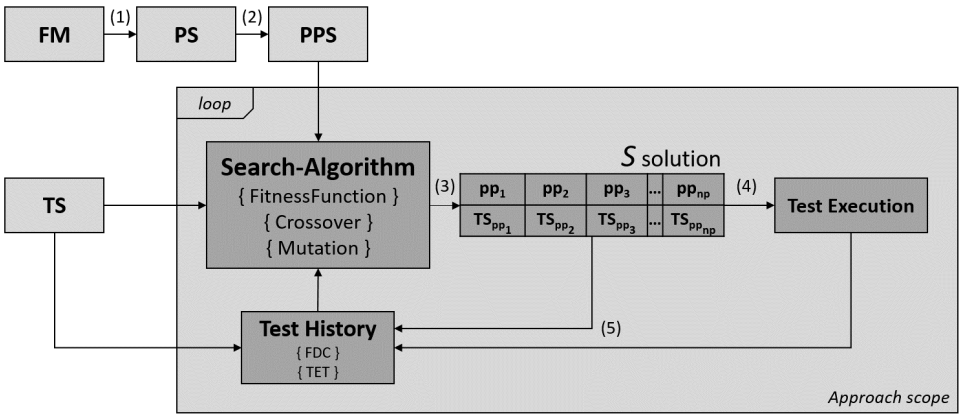


Figure 5.2: The proposed approach follows 5 steps: (1) Product selection [PSK+10, OZML11, MGSH13, TAK+14], (2) Product prioritization[LHFC+14, SSRC14a, AHTM+14, PSS+16, AHTL+16], (3) search algorithm execution and solution generation, (4) solution testing and (5) solution and test results saving. Steps 3-4-5 are repeated iteratively. Approach scope is limited by the frame.

The first step is the product selection which takes the Feature Model (FM) as an input, and generates the Product Suite (PS). FM represents all SPL product configurations. In this step the SPL product set is minimized to obtain the PS, based on any state-of-the-art approach [PSK+10, OZML11, MGSH13, TAK+14].

The second step involves the product prioritization, where all products of the PS are sorted according to any usual product prioritization technique [LHFC+14, SSRC14a, AHTM+14, PSS+16, AHTL+16]. As a consequence, those products with the best ability to reveal faults are tested first. The sorted Prioritized Product Suite (PPS) and the PS have an identical number of products np.

Before describing the third step, Test Suite (TS) and Test History should be explained. The TS accommodates a collection of test cases for testing PS, where each test case tests a specific part of a product at the application engineering level. Test History is a data storing mechanism that gathers test execution results for each test case and product combination, recording the Test

Execution Time (TET) and updating the Fault Detection Capability (FDC) of each test case. When there is no available real FDC data, the fault revealing capability of each test case can be estimated using mutation testing. Otherwise, preparatory iterations will be needed until required data is gathered.

In the third step, the proposed search algorithm is executed. This search algorithm takes PPS, TS and Test History as inputs, and once the algorithm is executed, it returns a solution. The generated solution consists of a set of allocated test cases for each of the prioritized products under test. Notice that each of the np number of products includes their own test suite TS_{np} based on the search. Each test suite has a maximum number of test cases (maxTC), which is predefined by the test engineer. When the algorithm performs the allocation, it is possible not to fill the test suite completely in order to provide the optimal solution. A sample solution is depicted in Figure 5.3, where empty slots of the test suite are represented with *null* test cases.

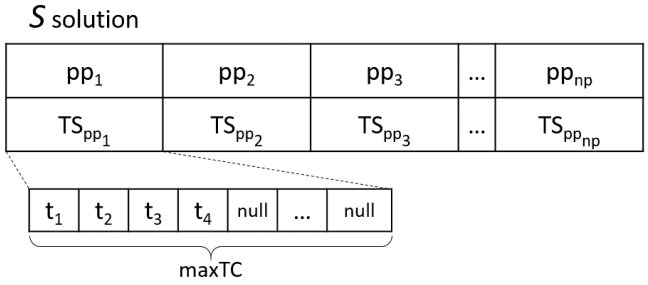


Figure 5.3: A solution generated for np products with $maxTC$ test cases allocated per test suite.

The fourth step consists of executing test cases of the solution. Finally, in the fifth step, generated solutions and results of test cases execution are saved into the Test History. This data is employed with two objectives. Firstly, it is used to update information related to the fault revealing capabilities of the executed test cases, so that the algorithms can guide the search more accurately in future runs. The second reason is to avoid allocating previously tested product and test case combinations during the following iterations. The last three steps (search algorithm execution, test execution and historical data saving) are iteratively repeated, until the test time budget is consumed. To implement the described steps of the approach an algorithm has been designed shown in Algorithm 1.

Algorithm 1: Iterative test allocation approach

```
1 system initialization;
2 PS = GenerateProducts(FM, GenerationCriterion);
3 PPS = PrioritizeProducts(PS,FM, PrioritizationCriterion);
4 while !Test Time Budget Consumed do
5     S = AllocateTestCases(TestHistory, MaxTC);
6     foreach Product under test do
7         TestResults = ExecuteTest(S);
8     UpdateTestHistory(TestResults);
```

5.3.1 Search algorithms measures

To allocate test cases cost-effectively in each of the products under test, three measures have been specified when defining the fitness function. It is important to highlight that these measures act at the test suite level and thus, the fitness function is calculated for each of the products under test in PPS.

Fault Detection Capability (FDC)

The first defined measure has been the Fault Detection Capability (FDC). This measure allocates effective test cases in terms of the ability to detect faults. It has typically been selected in product line engineering for other optimization tasks, such as test selection [AWSE16a], test suite minimization [WAG13, WAG15] and test prioritization [WBA+14, AWSE16b]. We calculate the FDC following Equation 5.1, where $maxTC$ is the number of test cases of TS_{pp_i} and SR_{tc_i} is the success rate.

$$FDC_{TS_{pp_i}} = \frac{\sum_{i=1}^{maxTC} SR_{tc_i}}{maxTC} \tag{5.1}$$

The success rate is updated every time a test case is run. A test case is considered successful if at least one fault is detected. Otherwise, the test case is considered to have failed. The success rate is computed this way because it is difficult in many contexts to differentiate between faults, as explained in [AWSE16b]. Thus, the success rate SR of a given test case tc_i is expressed in Equation 5.2 and calculated by the division of successful executions $NumSuc_{tc_i}$ with the total number of executions ($NumSuc_{tc_i} + NumFail_{tc_i}$).

$$SR_{tc_i} = \frac{NumSuc_{tc_i}}{NumSuc_{tc_i} + NumFail_{tc_i}} \tag{5.2}$$

Test Execution Time (TET)

It is the cost measure that quantifies the effort of executing the test cases. We show the TET calculation in Equation 5.3, where $norm(ET_{tc_i})$ is the normalized time required to execute tc_i test case. Weight-based algorithms require objective values to range between 0 and 1 [WAG13]. Thus, to avoid objective values out of this range the normalized time is used. The $norm(ET_{tc_i})$ is calculated dividing the execution time of each test case (i.e.,ET_{tc_i}) with the execution time of the longest test case in the test suite, coined as $maxTE$.

$$\begin{aligned} TET_{TS_{pp_i}} &= \frac{\sum_{i=1}^{maxTC} norm(ET_{tc_i})}{maxTC}; \\ norm(ET_{tc_i}) &= \frac{ET_{tc_i}}{maxTE}; \end{aligned} \tag{5.3}$$

Test Case Appearance Frequency (TCAF)

Selecting different test cases improves the detection of different faults. In our context, we select those test cases that have appeared less frequently. Thus, we measure the test case appearance frequency at test suite level $TCAF_{TS_i}$, trying to select test cases not previously used in the solution. To this end, as it is shown in Equation 5.4, we consider the times each test case of the current test suite has been previously used ($repTC$) and divide this by the maximum test suite size ($maxTC$).

$$TCAF_{TS_i} = \frac{repTC}{maxTC} \tag{5.4}$$

5.3.2 Fitness Function

Weight-Based Search Algorithms (WBSAs) convert a multi-objective problem into a single-objective one by assigning a specific weight to each of the search objectives [WAG15, AWSE16b]. Given that a search problem has N numbers of objective functions, each objective (Obj) is assigned a specific weight (w). Notice that the fitness function must range between the values 0 and 1. Consequently, each of the objectives should range between 0 and 1, and the sum of all the weights that are assigned to each objective must result in 1 (i.e., $\sum_{i=1}^{N} w_i = 1$). The generic fitness function for WBSAs is expressed in Equation 5.5.

$$FitnessFunction = \sum_{i=1}^{N} w_i \cdot Obj_i \tag{5.5}$$

WBSAs include several advantages. One such advantages is that they support user-preference, which is highly valued in testing to give higher importance to some measures (e.g., fault revealing capability can be more valuable than coverage when testing a specific system). Another advantage of WBSAs is that they can be easily integrated in any search algorithm, either a local (e.g., Greedy or the Alternating Variable Method) or a global search algorithm (e.g., Genetic Algorithms).

We took advantage of WBSAs theory to define our fitness function at two levels to guide search: (1) the application engineering fitness function and (2) the domain engineering fitness function. The former is related with the test suite assigned to each of the products under test, and thus, it is calculated individually for each of the products. The latter is related with all the products, which takes into account individually each of the application engineering level fitness functions and assigns weights to each of them based on the priorities of their products (i.e., position of the products in PPS).

Application engineering fitness function

The application engineering fitness function measures how good the allocated test cases in a specific product are in order to test this product cost-effectively. As a result, the three measures defined in Section 5.3.1 were employed. Following the WBSA theory, we assigned to each of the measures a specific weight. The higher this weight, the higher the relevance of the measure. This allows the test engineer to select the test preference by giving more or less relevance to each of the weights.

The particular fitness function f_i of a given product p_i is calculated following Equation 5.6. FDC_{p_i}, TET_{p_i} and $TCAF_{p_i}$ are specific normalized measure values of the p_i product and w_{fdc}, w_{tet} and w_{tcaf} are the weight values set up for this particular product, with the sum of weights to equal 1.

$$\begin{gathered} f_{p_i} = w_{fdc_{p_i}} \cdot (1 - FDC_{p_i}) + w_{tet_{p_i}} \cdot TET_{p_i} + w_{tcaf_{p_i}} \cdot TCAF_{p_i} \\ w_{fdc_{p_i}} + w_{tet_{p_i}} + w_{tcaf_{p_i}} = 1; \end{gathered} \tag{5.6}$$

Domain engineering fitness function

Once application engineering fitness functions are calculated, the domain engineering fitness function takes advantage of WBSAs to obtain a global fitness function. In this case, a specific weight is assigned to each of the

application engineering fitness functions. The domain engineering fitness function F is expressed in Equation 5.7.

$$F = \sum_{i=1}^{np} W_i \cdot f_{p_i} \tag{5.7}$$

In addition, each weight is different for each of the application engineering fitness functions in order to give more importance to those which contain a product with a higher priority according to PPS. This way, the domain engineering fitness function is more sensitive to those application engineering fitness functions where their product has more probability of containing a fault. This is performed as proposed in Equation 5.8, where, W_{p_i} is the specific weight to be calculated for a particular p_i product, $p_{i_{pos}}$ is the position of the product in the sorted PPS list and Z is a constant value, common for all products, and calculated summing the unitary fractions of the position of all PPS products.

$$W_{p_i} = \frac{1}{p_{i_{pos}} \cdot Z} \;\; ; \quad \longrightarrow \quad Z = \sum_{i=1}^{np} \frac{1}{i} \tag{5.8}$$

5.3.3 Crossover operator

The crossover operator we have implemented exchanges product test suites between two different solutions (parents) to create new solutions (children). This exchange is performed at the test suite level. Specifically, we implemented a single point crossover operator that randomly selects the crossover point value within the range $[1, np-1]$, np being the total number of products in PS. Subsequently, both solutions are combined taking into account the crossover point in order to create two children solutions. Figure 5.4 depicts an example of the implemented crossover operator for a solution with 3 products and a maximum 5 test cases when the crossover point is 2.

5.3.4 Mutation operator

We have implemented three different constrained sub-mutation operators as a mutation operator:

Addition sub-mutation operator

This sub-mutation operator is only available when the variable that must be mutated does not contain any test case (i.e., it is *null*). When this happens, a randomly chosen test case is added to the test suite.

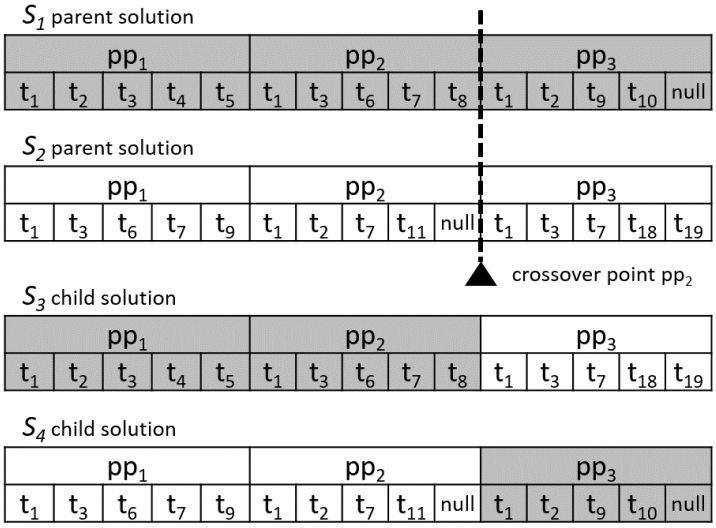


Figure 5.4: Crossover operator applied to simplified two parent solutions at crossover point = 2, generating 2 children solutions with test suites exchanged.

Removal sub-mutation operator

This operator is available when the variable to be mutated contains a test case (i.e., it is not *null*). Thus, the test case can be removed from the test suite. This sub-mutation operator has been defined for those cases where there are many test cases with a low probability of finding faults, or where there are very long test cases.

Replace sub-mutation operator

As in the previous sub-mutation operator, this operator is available when the variable to be mutated contains a test case. Thus, the variable can be randomly replaced by another test case. This sub-mutation operator has the objective of adding cost-effective test cases in the test suite.

Constraints handling

In addition, it is important to highlight that the first and third sub-mutation operators are constrained. The first constraint is that the sub-mutation operators cannot add test cases that have already been assigned in the current test suite. The second constraint is that they cannot add test cases that have already been allocated to the same products in previous iterations. To this end, all the previously obtained solutions are pre-processed by the algorithm and

the developed mutation operator takes them into account. Moreover, there are test cases that are not compatible with specific products. Thus, a third constraint is defined so that these test cases are not selected with incompatible products. Mutation operator samples are shown in Figure 5.5, where a parent solution is mutated by addition, removal and replace sub-operators.

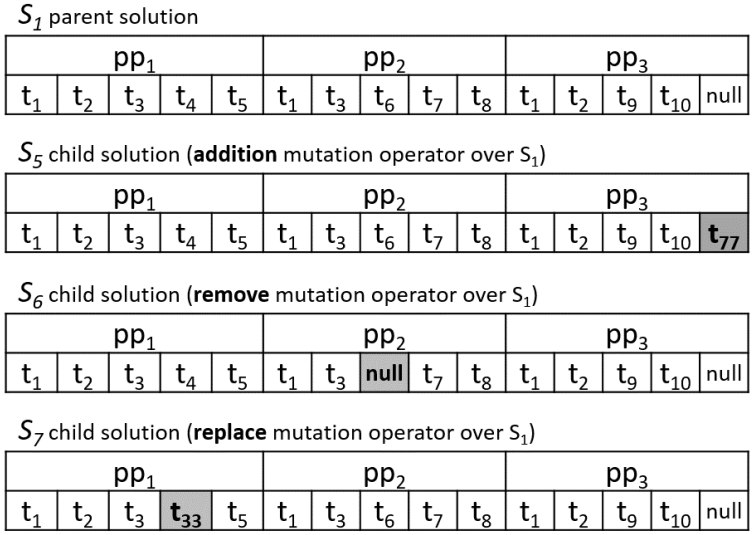


Figure 5.5: Mutation operators applied to simplified parent solution, generating 3 children solutions: (i) t_{77} added at *null* element position in S_5, (ii) t_6 removed leaving *null* in S_5, (iii) t_4 replaced with t_{33} in S_7.

5.4 Evaluation

5.4.1 Research Questions

The experiment's objective consists of evaluating the performance of the proposed algorithms when detecting faults. The first Research Question (RQ1) examines the proposed algorithms are effective compared with both Random Search (RS) and Traditional Testing Approach (TRA). The comparison between proposed algorithms and the RS, which is considered as baseline algorithm, is performed in order to assess that the problem to be solved is not trivial. The comparison with TRA aims to analyze the performance of the proposed algorithms within the traditional approach. The second Research Question (RQ2) measures the performance of the proposed algorithms. Finally, the third Research Question (RQ3) examines the scalability of the proposed algorithms by increasing the test suite size.

- **RQ1. Sanity Check:** Are the proposed search algorithms effective when compared with existing reference algorithms? We divide RQ1 into the following sub-Research Questions:
  - **RQ1.1 RS baseline:** Are the selected search algorithms effective when compared with RS?
  - **RQ1.2 TRA reference:** Are the selected search algorithms effective when compared with TRA?
- **RQ2. Performance:** Which of the selected algorithm configuration fares best among those selected?
- **RQ3. Scalability:** How does the test suite size affect the performance of the selected search algorithms?

5.4.2 Experiment design and setup

Case study

We selected the Unmanned Aerial Vehicle (UAV) case study for the experimentation. This case study is further detailed in Section 4.4.2 and was based on a quad-copter Highly-Configurable Cyber-Physical System (HCCPS) employed in other evaluations by Arrieta et al. [AWSE16a, ASEZ16, AWM+17b]. Arrieta et al. extended another previous work published by Mosterman et al. [MSB+14] by adding variability to transform the system into a product line. Specifically, this product line was composed by 46 different features in order to describe the variability of both software and hardware characteristics and functionalities.[1] The UAV was modeled in MATLAB/Simulink containing 843 simulink blocks distributed in 4 depth levels. Additionally, a Test Suite of 120 test cases was developed based on domain knowledge [AWSE16a]. In our context, a test case consists of a set of signals that stimulates the system's inputs. Besides, any test case can be applied to any product. However, any state-of-the-art technique can be employed when generating test cases [AWM+17b].

[1] A detailed overview of the system can be found online in the following web page: https://sites.google.com/a/mondragon.edu/asterysco/case-study

The product line could generate more than 800,000 products. When applying pair-wise based selection 22 products were generated and, when applying 3-wise selection 85 products were generated. Thoroughly testing products generated by pair-wise will last an estimated 848 hours, while testing products generated by 3-wise will last an estimated 4,046 hours.

The PS was derived with the ICPL algorithm following the pair-wise product selection criterion [JHF12]. The PS was prioritized taking into account the variability coverage and cyclomatic complexity (VC&CC) criterion [SSRC14a] as it gave the best results for detecting faults in SPL test case prioritization. Although our approach provides an iterative technique to allocate test cases, we performed the experiment with one iteration to measure how good the approach was at detecting faults in the first round.

Artificial problems

Six different TS sizes (70, 80, 90, 100, 110, 120) were employed to evaluate the approach under different scenarios and to assess the scalability of the approach. In addition, four maxTC values (1, 2, 5 and 10) were employed after launching preliminary experiments and analyzing how different maxTC values performed. The combination of the six TS sizes and the four maxTC values, generated 24 artificial problems. Each artificial problem was combined with each of the selected algorithms. Each algorithm was run 100 times to account for random variations, as recommended by Arcuri and Briand [AB11].

Algorithms

We evaluated two algorithms each with two different fitness functions. One of the selected algorithms was the Weight-Based Genetic Algorithm (WBGA), which uses fixed weights in the weighted fitness function. The second algorithm was the Random-Weighted Genetic Algorithm (RWGA), which, unlike WBGA, assigns the weights to the objectives randomly in each iteration. We selected these algorithms based on their previous good performance in similar problems [WAG15, WAG13]. In addition, we created two variants of the fitness function described in the approach. The first fitness function (f1) considered TET and FDC, while the second one considered TET, FDC and TCAF. The combination of two search algorithms (WBGA, RWGA) and two fitness functions (f1,f2) produced the four **proposed algorithms** analyzed in this experiment: $WBGA\_f1$, $WBGA\_f2$, $RWGA\_f1$ and $RWGA\_f2$.

Before evaluating the results of the proposed algorithms, we selected Random Search (RS) algorithm and traditional approach based algorithms (TRA) as references for comparison. RS is commonly used [WAG15, PSS+16] as a comparison baseline in order to assess that the problem to be solved is not trivial. TRA was based on our previous work [AWSE16a], where WBGAs are employed to select every test case for each individual product under test. Both RS and TRA algorithms used the defined fitness functions (f1, f2), producing four **reference algorithms**: RS_f1, RS_f2, TRA_f1, TRA_f2.

Specific weights for each measure were assigned to configure the fitness functions of each algorithm according to figures shown in Table 5.1. For fixed weighted algorithms, the weight assigned provided identical importance to each measure. For random weighted algorithms, weights were randomly assigned ensuring the sum of weights was equal to 1.

Comparisons

To address RQ1.1 we compared the performance of the proposed algorithms with RS baseline algorithms for the defined artificial problems. Similarly, to address RQ1.2 we compared the proposed algorithms with TRA algorithms. To address RQ2 we compared all possible combinations of the proposed algorithms among themselves with the defined artificial problems. To address RQ3 we evaluated the algorithms with different test suite sizes (from 70 to 120 with an increment of 10 test cases).

Algorithms parameters configuration

For the parameters of the GAs, (both from our approach and the ones related to TRA), we set the crossover probability to the 0.8 value. The mutation probability was set to 1/N, with N being the number of variables in a solution (in our case the number of products np multiplied by the $maxTC$ parameter). The population size was set to 100 and the number of generations to 1000. Thus we considered a total amount of 100,000 fitness evaluations.

Evaluation metrics

We employed mutation testing to evaluate our approach since it has been demonstrated that it is a good substitute for real faults [JJI+14]. We automatically generated 20 mutants based on the tool used in previous works [AME17]. These mutants consist of variants of the UAV solution with faults injected

Table 5.1: Measure weights for fitness function configuration

Algorithm	Measures		
	TET	FDC	TCAF
RS_f1	0.5	0.5	0
RS_f2	1/3	1/3	1/3
TRA_f1	0.5	0.5	0
TRA_f2	1/3	1/3	1/3
WBGA_f1	0.5	0.5	0
RWGA_f1	random	random	0
WBGA_f2	1/3	1/3	1/3
RWGA_f2	random	random	random

into MATLAB/Simulink blocks. Due to the complex mathematical models of the physical processes, mutation testing with MATLAB/Simulink becomes a costly computational activity when running simulation. In other similar works a likely number of mutants were employed (e.g., 13,17,21,44) [MNBB16], These injected faults were related to individual features of the FM and thus, the mutant is removed in case the product does not include the feature associated to the mutant. The distribution of faults was 50% in the physical layer and 50% in the cyber layer (i.e., software of the system).

Two metrics were defined to evaluate the generated solutions. The first metric used is named **Mutation Score** and measures the number of detected mutants in the first iteration. The second metric used is named **Average Fault Detection Time (AFDT)** and measures the required mean time to detect each mutant in the first iteration.

5.4.3 Results

To statistically evaluate both RQ1 and RQ2, we divided the results into 24 data sets corresponding to the defined artificial problems. With each data set, we first refused the null hypothesis for the normality distribution of data by the Sapiro-Wilk test [WAGL15]. We then applied the Vargha and Delaney $\hat{A}_{12}$ and Mann-Whitney U tests following the guidelines proposed by Arcuri et al. [AB11]. In addition, we set the statistical significance to 95 % (i.e., there is a statistical significance if $p - value < 0.05$). The results presented in Tables 5.2 and 5.3 show the number of artificial problems that each algorithm had significantly better than the other for each algorithm pair. They also show the number of artificial problems where there was no statistical significance

between the performance of each algorithms. To evaluate RQ3, we measured the correlation between the performance of the algorithms with respect to the number of test cases using the Spearman's rank correlation test.

Table 5.2 illustrates results for RQ1 while RQ2 is presented in Table 5.3. RQ3 results are shown in Table 5.4. Both Tables 5.2 and 5.3 are presented with similar structures due to both comparing the performance of algorithms taken in pairs, after applying the Vargha and Delaney $\hat{A}_{12}$ statistics and Man-Whitney U test. For each of the evaluation metrics, 3 columns are reported. The column $A > B$ presents the number of artificial problems for which the algorithm in column A showed statistically significantly better performance than the algorithm B (i.e., $p - value < 0.05$ and $\hat{A}_{12}$ in favor of the algorithm A). $B > A$ represents the opposite. The column $A = B$ reports the number of artificial problems for which there was no statistically significant difference between the algorithms A and B (i.e., $p - value > 0.05$).

Table 5.2 compares proposed algorithms $WBGA\_f1$, $WBGA\_f2$, $RWGA\_f1$ and $RWGA\_f2$ with defined reference algorithms RS_f1, RS_f2, TRA_f1 and TRA_f2 for both Mutation Score and AFDT metrics. As for RQ1.1 results reflect that proposed algorithms outperformed RS algorithms in 3 out of 4 comparisons for the Mutation Score metric as it is shown in Table 5.2. As for the AFDT metric, the selected algorithms outperformed RS from 20 to 22 artificial problems out of 24. Consequently, we can conclude that the proposed algorithms outperformed RS baseline and thus, the problem to solve is non-trivial.

Similarly, regarding RQ1.2 results reflect that proposed algorithms outperformed the traditional testing in 3 out of 4 comparisons for the Mutation Score metric, as it is shown in Table 5.2. As for the AFDT metric, the selected algorithms outperformed the traditional testing in all four comparison. Therefore, we can conclude that the proposed algorithms significantly outperformed the traditional testing approach (based on GA-based TC minimization).

Table 5.3 compares the performance of the proposed algorithms. Results for the Mutation Score metric show that the RWGA algorithms performed better than WBGA, no matter which fitness function was employed. As shown in the last two rows, we can conclude that the algorithms using the TCAF measure (implemented in those algorithms ended with the "$\_f2$" suffix) improved performance results for this metric, with $WBGA\_f2$ and $RWGA\_f2$ results better than $WBGA\_f1$ and $RWGA\_f1$ respectively.

As for the AFDT, it can be observed in Table 5.3 that WBGA algorithms

Table 5.2: Summary of the Vargha and Delaney $\hat{A}_{12}$ statistics and Man-Whitney U test for RQ1

RQ	Algorithm		Mutation Score			AFDT		
	A	B	A >B	A=B	B >A	A >B	A=B	B >A
1.1	WBGA_f1	RS_f1	14	1	9	21	2	1
1.1	RWGA_f1	RS_f1	11	3	10	21	2	1
1.1	WBGA_f2	RS_f2	12	3	9	22	2	0
1.1	RWGA_f2	RS_f2	10	3	11	20	3	1
1.2	WBGA_f1	TRA_f1	3	12	9	21	0	3
1.2	RWGA_f1	TRA_f1	9	13	2	19	2	3
1.2	WBGA_f2	TRA_f2	11	11	2	17	1	6
1.2	RWGA_f2	TRA_f2	12	9	3	14	1	9

performed in general better than RWGA algorithms, as the number of artificial problems with better performance was significantly higher. From the last two rows, we can conclude that the algorithms not considering the TCAF measure (implemented in those algorithms ended with the "$\_f1$" suffix) improved AFDT performance results.

Table 5.3: Summary of the Vargha and Delaney $\hat{A}_{12}$ statistics and Man-Whitney U test for RQ2

Algorithm		Mutation Score			AFDT		
A	B	A >B	A=B	B >A	A >B	A=B	B >A
WBGA_f1	RWGA_f1	2	13	9	16	5	3
WBGA_f1	RWGA_f2	6	9	9	21	3	0
WBGA_f2	RWGA_f1	3	15	6	6	3	15
WBGA_f2	RWGA_f2	3	14	7	16	7	1
WBGA_f1	WBGA_f2	5	10	9	19	2	3
RWGA_f1	RWGA_f2	2	13	9	21	3	0

Table 5.4 presents the results of the Mutation Score and AFDT metrics for RQ3. The first two columns identify the metric and the algorithm analyzed. The following two columns report (1) the Spearman's rank correlation (ρ), followed by (2) the statistical significance $Prob > |\rho|$. These two columns are repeated for each of the 4 maxTC values analyzed.

As shown in Table 5.4, the correlation for the Mutation Score is positive, meaning that as the number of test cases increases, the number of detected faults increased. In Table 5.4, the correlation is negative for most of the cases

(13 out of 16), meaning that as the number of test cases increased, the AFDT decreased. Thus, we can conclude that our approach scales up to a more complex problem in terms of number of test cases.

Table 5.4: Summary of Spearman's rank correlation test for Mutation Score (MS) and AFDT metric results in RQ3

Metric	Algorithm	maxTC = 1		maxTC = 2		maxTC = 5		maxTC = 10	
		ρ	$Prob > \|\rho\|$	ρ	$Prob > \|\rho\|$	ρ	$Prob > \|\rho\|$	ρ	$Prob > \|\rho\|$
MS	WBGA_f1	-	-	0.878	<0.0001	0.855	<0.0001	0.839	<0.0001
	WBGA_f2	0.124	0.002	0.622	<0.0001	0.754	<0.0001	0.845	<0.0001
	RWGA_f1	0.104	0.011	0.577	<0.0001	0.673	<0.0001	0.625	<0.0001
	RWGA_f2	0.115	0.005	0.419	<0.0001	0.570	<0.0001	0.574	<0.0001
AFDT	WBGA_f1	-0.885	<0.0001	-0.630	<0.0001	0.209	<0.0001	0.262	<0.0001
	WBGA_f2	-0.868	<0.0001	-0.823	<0.0001	-0.655	<0.0001	0.485	<0.0001
	RWGA_f1	-0.869	<0.0001	-0.735	<0.0001	-0.452	<0.0001	-0.432	<0.0001
	RWGA_f2	-0.855	<0.0001	-0.755	<0.0001	-0.552	<0.0001	-0.433	<0.0001

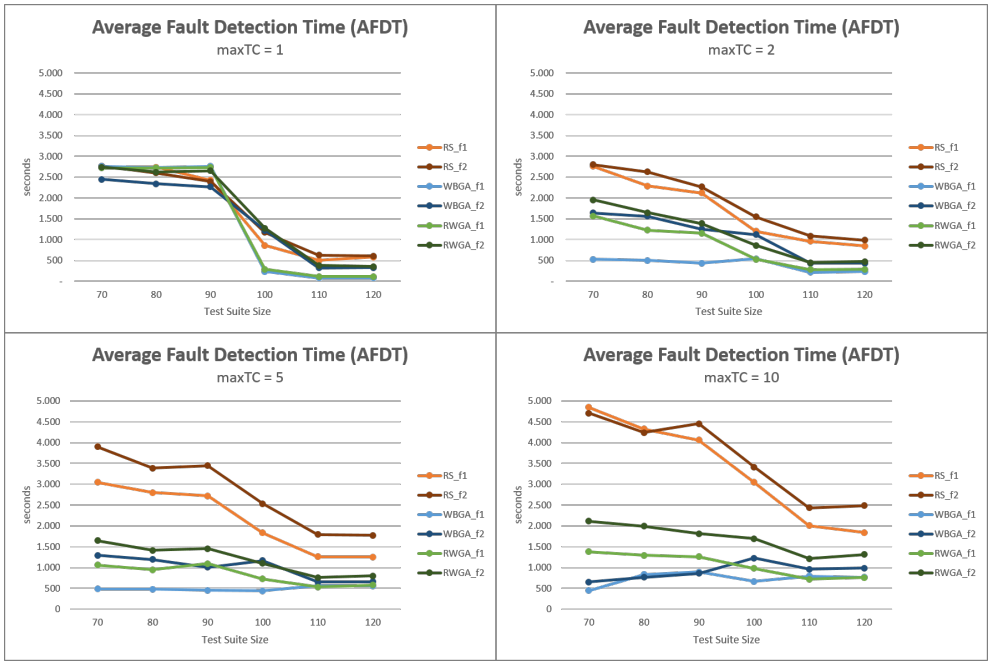


Figure 5.6: AFDT metric charts comparing mean values of each artificial problem per *maxTC*.

5.4.4 Discussion

The general objective of RQ1 was to evaluate if the proposed algorithms improved the selected reference algorithms. This was carried out by two sub-research questions. RQ1.1 aimed at assessing whether the proposed problem was non-trivial to solve by comparing the proposed algorithms with RS.

> *For most of the evaluation metrics the proposed algorithms were significantly better than RS.*

The performance obtained for the AFDT metric by the proposed algorithms was notable. The graph shown in Figure 5.6 presents the AFDT metric charts comparing mean values of each artificial problem per maxTC. Here it can be outlined that for the AFDT metric, the proposed algorithms outperformed the RS algorithms on average in 39%, 74%, 73% and 75% for each of the defined *maxTC* (1,2,5,10) respectively. However, in some cases RS showed better performance at finding mutants. This might be due to the fact that with short test cases, our algorithms gave more importance to the test execution time rather than to the FDC. However, this problem could be solved in the future by assigning higher weights to the FDC measure for those cases where the test suite has fewer test cases. RQ1.2 aimed at comparing the proposed approach with the current practice of testing each product one by one with a test suite minimization algorithm based on GA, as proposed in [WAG15, WAG13, AWSE16a].

> *Results reflected that for the AFDT metric, the proposed algorithms outperformed the TRA algorithms on average in 23%, 79%, 75% and 64% for each of the defined maxTC respectively.*

RQ2 was established to compare the performance of the selected algorithms among them and based on the obtained results two conclusions can be highlighted:

> *On the one hand, results for the Mutation Score metric reflected that RWGAs showed better performance than WBGAs. On the other hand, results for the AFDT metric showed that WBGAs presented significantly better performance than RWGAs, particularly those algorithms which did not consider the test case appearance frequency measure (TCAF).*

RQ3 aimed at evaluating whether the test suite size affects the performance of the proposed algorithms.

> *It can be outlined that as the test suite size increases, the performance of the algorithms is better.*

The reason for this might be that making the test suite larger, the diversity of test cases is increased, incrementing the chances to build a better test suite for each of the product and improving the possibility to detect faults.

Concluding Remark

The experiment has employed only one case study, for that reason results should not be generalized. However, results obtained in this specific experiment conclude that the selected algorithms are better than RS, and thus, the problem to solve is non-trivial. In addition, our approach of iteratively testing the selected case study (which is a CPS product line) has outperformed the traditional approach of thoroughly testing each of the products one by one, in terms of faults detection time.

Based on the performed experiments, we propose to use the $RWGA$ with the second fitness function (i.e., the one that uses the FDC, TET and TCAF) if the number of test cases in the test suite is small. This way, the probability of finding faults is higher, despite the fact that the average time to detect them can be larger than the rest. With large test cases, we recommend using $WBGA\_f1$, since with large test cases the number of detected faults increases, and this algorithm is the fastest one to detect them. In addition, based on the results from Figure 5.6, we recommend to set the $maxTC$ size to 2 or 5 if the number of test cases in the test suite is less than 100. For TS sizes of 100 or more test cases , we recommend to set the $maxTC$ size to 1.

5.4.5 Threats to validity

This section identifies threats that could invalidate the performed empirical evaluation. An **internal validity** threat could arise due to the algorithms parameter configurations remaining constant during the entire experiment. Different algorithm configurations might lead to different results [AF11]. However, the assigned settings are in accordance with the guidelines from the literature related to search-based software engineering [AB11]. The experiment employs 20 mutants and, considering that systems might have more faults,

a second **internal validity** threat has been identified. In addition, another threat of mutation testing involves subsumed mutants [PHH+16]. To minimize the impact of these threats, different types of faults were introduced across the system. A **conclusion validity** threat could possibly arise due to the random variations of search algorithms. To mitigate this threat each algorithm is repeated 100 times and results are statistically tested following the guidelines proposed in the literature [AB11]. The **construct validity** threat relates to the comparison measures of all algorithms. To reduce this threat, we used the same stopping criterion for all the algorithms (i.e., 100,000 fitness evaluations). The **external validity** threat refers to the number of case studies used. At this point, the experiment has employed only one case study and results should not be generalized. However to minimize this threat the selected case study size is noteworthy with 843 Simulink blocks distributed over 4 depth levels. Moreover, we divided the evaluation in 24 artificial problems.

5.5 Related Work

SPL testing challenges have been extensively discussed [ER11, NdCMM+11, dCMMCDA14], however recent surveys and mapping studies reveal a renewed interest [HJK+14, LHFC+16]. According to the software engineering framework defined by Pohl et al. [PBL05], SPL testing sub-processes are presented at Domain Engineering and Application Engineering levels. Focusing on Domain Testing, relevant systematic studies [LHFRE15, TAK+14] have identified that Combinatorial Interaction Testing (CIT) is the leading selection approach for testing in SPLs. Furthermore, according to Thüm et al. [TAK+14] most of the research and industry communities have adopted the use of pairwise testing [POS+12] to select products from variability models from different techniques for CIT. In addition to the product selection problem, the product prioritization problem has been addressed by multiple authors. Search-based algorithms have been proposed by some authors [HPP+14, PSS+16]. There are other works that have proposed the use of heuristics [SSRC14a, SSPRC15] to prioritize products once they have been generated, while other authors prioritize products before generating them based on dissimilarity metrics [AHTM+14, AHTL+16]. An additional systematic mapping study conducted by Lopez-Herrejon et al. on Search-Based Software Engineering (SBSE) for SPLs [LHLE15] is also remarkable. They conclude that the most common SBSE application is for testing at the Domain Engineering level, (i.e., computing test suites in CIT)

and the most common techniques use genetic algorithms and multi-objective evolutionary algorithms [POS+12]. Recent works have been published in order to research at the Application Engineering level. Wang et al. applied a weighted genetic algorithm to minimize SPL test suites, while retaining fault detecting ability [WAG13, WAG15]. They proposed an additional systematic and automated methodology to manage test case selection based on feature models [WAGL15].

Incremental approaches have recently proposed to cope with different SPLs challenges. A risk-based testing approach for SPL integration testing has recently been proposed [LBL+17] to incrementally test SPLs. Failure probabilities and impacts are automatically computed for each of the selected variants, prioritizing important changes. Subsequently, a product ordering optimization incremental approach has recently been proposed [LAHTS17] based on graph-algorithms and existing heuristics.

To the best of our knowledge, most papers focus on generating relevant products for testing SPLs by processing a variability model. In addition, those works that focus on product prioritization [AHTM+14, AHTL+16, SSRC14a, SSPRC15], do not usually consider the time required by each of the products to be tested (i.e., they do not take application engineering level test cases). Our approach, proposes a search-based algorithm to perform the testing of product lines by allocating a small number of test cases to each of the products. This approach is performed iteratively, focusing on fault detection time minimization and combining well known Domain Testing techniques with proposed Application Testing search-algorithms.

5.6 Conclusions

This paper first proposes an approach that performs both the product selection and prioritization following well known techniques. Then we propose a novel approach to allocate a small number of test cases to each product in order to iteratively test the product line. To this end two fitness function levels have been implemented to guide the search. We empirically evaluated four proposed search algorithms with the goal of identifying the best algorithm for optimizing fault detection time. Results showed that RWGAs combined with appearance frequency objectives presented the best performance to detect faults (Mutation Score), while WBGAs provided the best results in terms of the required time to detect faults (AFDT). Furthermore, the average time to

detect faults of the proposed algorithms have outperformed RS algorithms in on average 65%. Similarly, the results of our algorithms have outperformed traditional test approach algorithms in on average 61%. We can conclude that our approach has significantly improved the performance of both metrics when compared with RS and traditional testing approach.

The results obtained by the proposed approach allowed us to empirically validate the effectiveness of using a small number of test cases in a iterative manner when testing HCCPSs. With this validation the first line of the exploratory work was concluded.

Chapter 6

Test Case Selection of HCCPS using Structural Coverage

Contents

This chapter provides details of the second exploratory research work. Section 6.1 contextualizes the presented contribution in the theoretical framework. Section 6.2 provides the technical motivation for this contribution. Section 6.3 presents our analysis proposal for test case selection. The approach is evaluated in Section 6.4. Section 6.5 positions our approach with the current literature. Finally, conclusions are summarized in Section 6.6.

6.1 Contribution overview

The research work presented in this chapter details the second exploratory work of the thesis. Figure 6.1 highlights this work (in blue) in the context of the entire theoretical framework (further details of the theoretical framework are provided in Section 4.3).

This exploratory work was set in order to validate the second hypothesis of the thesis, which aims at optimizing HCCPS testing by considering domain-level structural coverage information.

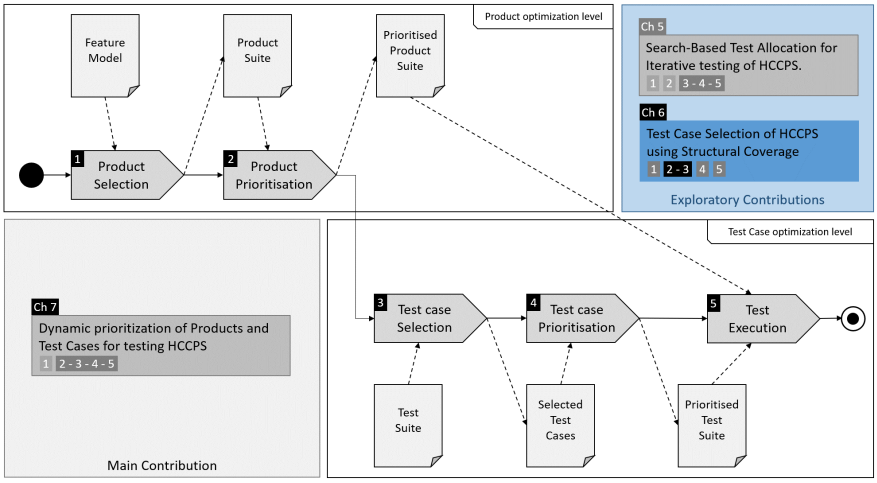


Figure 6.1: Second exploratory method proposed in the context of the theoretical framework. The second exploratory method "Test Case Selection of HCCPS using Structural Coverage" (blue colored), which involves steps number 2 and 3.

The approach proposed in this second exploratory work is designed to develop the *core concept* (defined at the theoretical framework in Section 4.3) by exploring how to consider product line information at both domain and Application levels. Specifically, it is focused on step number 3 of Figure 5.1, where a *study is carried out on analyzing how to use structural coverage*

when selecting test cases. To this end, we propose three different test case selection methods that consider a given time budget to test product lines in an efficient manner by means of structural coverage information. We analyze the three methods with three white-box coverage criteria (i.e., Decision Coverage, Condition Coverage and Modified Condition/Decision Coverage).

When analyzing the three test case selection methods, where Domain level information is considered, there is a modification of the pre-established ordering of the prioritized products. Specifically, when considering the product line coverage level information, one of the methods modifies the order of the prioritized products to optimize the testing. Therefore, in this contribution we also consider that step number 2 of Figure 5.1 is modified. Additionally, by merging these steps (i.e. 2 and 3) we set the basis for a later definition of pair concept (in the main contribution described in Chapter 7).

We evaluated the different test case selection methods with the CW case study which is further detailed in Section 4.4. The evaluation was carried out applying mutation testing technique. The results suggested that considering coverage information at the domain engineering level helps on detecting more faults, particularly when time budgets are low.

This second work allowed us to empirically validate the exploratory line of work proposed to optimize the testing of HCCPSs by considering the information at the Domain level and stating the basis for the future pair conception.

6.2 Introduction

Testing Software Product Lines (SPLs) is a tedious and time consuming activity. This is mainly caused by the large number of products that a SPL can be set to. This makes it impossible to test all valid product configurations. As a result, many approaches focus on deriving a subset of relevant products that are representative for testing the entire SPL [LHFRE15, dCMMCDA14, ER11]. This is typically done by following a combinatorial approach, in which the interaction of features is ensured.

Most SPL testing studies focus on the domain engineering level. Specifically, a large number of studies have focused on deriving a representative subset of products from a feature model [KBK11, POS+12, HPP+13b, LHFRE15], whereas others focus on prioritizing the yielded products [SSRC14a, PSS+16, AHTL+16, DPC+15, AHLL+17]. Nevertheless, there are some studies that

have focused on the application engineering level by selecting or minimizing a subset of *test cases*[1] for testing a specific product [SMP10, WAG13, WAG15, AWSE16a] or by prioritizing them [WBA+14, AWSE16b, LLL+15]. Most of these Application level testing approaches focus on regression testing by considering historical data of test cases (e.g., considering the number of faults revealed by each test case). Although these test case selection methods are effective, a drawback is that each of the test cases need to be executed several times to obtain valuable historical data. Therefore, historical data related to faults is not always available. This chapter focuses on test case selection criteria based on white-box coverage metrics for time constrained scenarios.

Many studies in software engineering have proposed the use of structural coverage to select or prioritize test cases [MB03, YH12, YH07]. However, the use of structural *coverage*[2] in the context of SPLs has centered little attention. In this work we propose different methods for test case selection based on structural coverage information. Specifically, we employ two metrics for structural coverage, proposing three different test case selection methods based on these metrics. Furthermore, we perform the analysis employing three different coverage criteria, i.e., Decision Coverage (DC), Condition Coverage (CC) and Modified Condition/Decision Coverage (MC/DC). We empirically evaluated these methods within a case study from the automotive domain with different time budgets. Results suggest that one of the methods, which focuses on selecting test cases considering the overall coverage of the SPL, performs better than the remaining methods at detecting faults when low time budget exist. As for the analyzed coverage criteria, results suggest that the three of them (i.e., DC, CC and MC/DC) obtain similar performance for the evaluated case study.

[1] In the context of SPLs, the term *test case* is employed both to refer a product configuration or a test exercising a product configuration. This work employs the latter one.

[2] In the context of SPL engineering, typically t-wise coverage is considered to derive products from a feature model following combinatorial interaction testing approaches [CDS08, POS+12]. However, this kind of coverage should not be confused with the concept of structural coverage.

6.3 Test Case Selection compared methods

We propose three test case selection methods considering structural coverage criteria for a time-constrained scenario.

6.3.1 Basic Concepts

Let $PS = \{P_1, P_2, ..., P_{np}\}$ be the Product Suite to be tested, np being the total number of products. This product suite can be derived by following any state-of-the-art product derivation approach (e.g., a combinatorial product derivation approach [OMR10, POS+12]). $TS_{P_i} = \{TC_{1P_i}, TC_{2P_i}, ..., TC_{ntcP_i}\}$ is the test suite with $ntcP_i$ number of test cases employed to test product P_i. This test suite is assigned to the specific product P_i and the test cases can be derived following any established approach (e.g., Adaptive Random Testing [CKMT10], or model-based testing [WL13]). Last, we consider the Product Line Test Suite, the union of all test suites (i.e., $PLTS = \{TS_{P_1}, TS_{P_2}, ..., TS_{P_{np}}\}$).

6.3.2 Motivating Example

As an example to illustrate concepts of this section, we selected the feature model of a SPL for mobile phones presented by Benavides et al. [BSC10]. Figure 6.2 depicts a simplified feature model from the mobile phone industry. The features are used to specify different mobile phone software configurations in products. Thus, all products must include call support, as well as any of the screen alternatives (i.e., basic, colour or high resolution screen type). Optionally GPS and multimedia devices (i.e., Camera, MP3 or both) might be included (further details provided in Section 2.2.2).

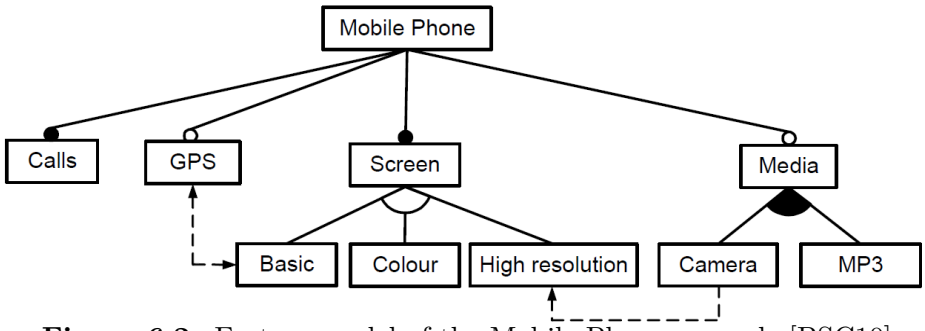


Figure 6.2: Feature model of the Mobile Phone example [BSC10].

The Mobile Phone feature model can provide fourteen valid product configurations. Let us consider three representative product configurations in

order to simplify the example. Therefore, the product suite is composed of the selected three products $PS = \{P_1, P_2, P_3\}$. The configured features for P_1 are Mobile Phone, Calls, Screen and Colour. Features for P_2 are Mobile Phone, Calls, Screen, High resolution, Media and Camera. Finally, features for P_3 are Mobile Phone, Calls, GPS, Screen, Basic, Media, MP3.

In Section 6.3.4, the description of the proposed test case selection methods is presented. Two of the three test case selection methods employ a product prioritization criteria, as different studies have shown that establishing a specific order of the products derived from the product line improves fault detection effectiveness [SSRC14a, AHTL+16]. Thus, products from the product suite of this example are also prioritized using any state of the art criterion [AHTL+16, SSRC14a]. Resulting prioritized product suite is $PS_{prioritized} = \{P_3, P_2, P_1\}$. Note that the order of the products has been modified compared to PS.

In addition, test cases are generated for each product, obtaining specific test suites per product. Thus, the test suite assigned to product P_1 is $TS_{P_1} = \{TC_{1P_1}, TC_{2P_1}, TC_{3P_1}\}$. The test suite assigned to product P_2 is $TS_{P_2} = \{TC_{1P_2}, TC_{2P_2}, TC_{3P_2}\}$, while the test suite assigned to product P_3 is $TS_{P_3} = \{TC_{1P_3}, TC_{2P_3}, TC_{3P_3}\}$.

To complete this example, the association between test cases and features is established. To this end, it should be clarified beforehand that typically each feature in a product line is related to a set of assets that implement the feature's functionality (e.g., code, test cases, documentation, etc.). Particularly in this example, each feature is related to a piece of code. The coverage of each piece of code is obtained by satisfying a number of test requirements. For instance, let us assume that the code block associated with the feature *Media* in Table 6.1 contains a decision (e.g., `if(z>8)` condition). The test requirement TR13 defines a test target to obtain the *true* outcome in the decision, while the test requirement TR14 is defined to obtain the outcome *false*. The detailed association between test cases and covered test requirements is shown in Table 6.1. Thus, when a test case satisfies a test requirement, a • is placed at the intersection. For example, test case TC_{3P_1} covers test requirements $TR1, TR8$ and $TR9$. Finally, note also that column TET provides required Test Execution Time (TET) of each test case in seconds.

Table 6.1: Test Requirement (TR) coverage and Test Execution Times (TET) per test case.

Products	Test Cases	Mobile Phone		Calls			GPS	Screen			Basic	Colour	High resolution	Media		Camera	MP3	TET
		TR1	TR2	TR3	TR4	TR5	TR6	TR7	TR8	TR9	TR10	TR11	TR12	TR13	TR14	TR15	TR16	
P1	TC_{1P_1}	•	•	•				•				•						2
	TC_{2P_1}	•			•	•		•										5
	TC_{3P_1}	•							•	•								3
P2	TC_{1P_2}	•			•	•		•					•					3
	TC_{2P_2}		•			•				•			•	•	•	•		8
	TC_{3P_2}	•		•	•	•		•	•	•			•					1
P3	TC_{1P_3}	•		•		•				•					•		•	4
	TC_{2P_3}	•	•	•	•		•	•	•		•			•			•	6
	TC_{3P_3}	•		•	•	•				•	•							2

6.3.3 Structural Coverage Levels

We employ the structural coverage metric at two different levels: on the one hand, the traditional use of structural coverage for individual products, named Product Structural Coverage (PSC). On the other hand, the structural coverage that considers the entire product line, named Product-Line Structural Coverage (PLSC).

Product Structural Coverage (PSC)

Given a product P_i, and a set of test cases testing this product, the structural coverage at the product level (PSC) for product P_i is the percentage of test requirements covered by the set of test cases. Referring to the example in Table 6.1, let us consider the following test suite of P_1: $TS'_{P_1} = \{TC_{1P_1}, TC_{3P_1}\}$. Test case TC_{1P_1} covers 5 test requirements (i.e., $TR1, TR2, TR3, TR7$ and $TR11$). Test case TC_{3P_1} provides 3 test requirements (i.e., $TR1, TR8$ and $TR9$) but only 2 new ones with respect to TC_{1P_1} (i.e., $TR8$ and $TR9$). The total number of test requirements covered by TC_{1P_1} and TC_{3P_1} is 7. P_1 has in total 9 test requirements associated to the features it has. Thus, the PSC of the test suite TS'_{P_1} is 7/9, meaning that more than 77% of the test requirements have been covered for P_1 when selecting the test cases TC_{1P_1} and TC_{3P_1}.

Product-Line Structural Coverage (PLSC)

The structural coverage at the product line level (PLSC) quantifies the number of covered test requirements with respect to the total number of different test requirements of the entire product line. Continuing with the example of this section, let us consider the next test suite $PLTS' = \{TC_{1P_1}, TC_{2P_2}, TC_{3P_3}\}$. The first test case (i.e., TC_{1P_1}) covers 5 test requirements (i.e., $TR1$, $TR2$, $TR3$, $TR7$ and $TR11$), the second test case (i.e., TC_{2P_2}) covers 7 test requirements

(i.e., $TR2$, $TR5$, $TR9$, $TR12$, $TR13$, $TR14$ and $TR15$), and the third test case (i.e., TC_{3P_3}) covers 6 test requirements (i.e., $TR1$, $TR3$, $TR4$, $TR5$, $TR9$ and $TR10$). Note however, that some of the test requirements are covered more than once by different test cases within different products (e.g., $TR1$ and $TR3$ are covered by TC_{1P_1} and TC_{3P_3}, while $TR2$ is covered by TC_{1P_1} and TC_{2P_2}). Thus, in total, at the product line level, 13 test requirements are covered. Since the entire product line consists of 16 test requirements, the coverage at the product line level when selecting these test cases will be 13/16, meaning that more than 81% of the product line test coverage has been covered.

6.3.4 Test Case Selection Methods

We propose three different methods for test case selection: The Application level Method (AM), the Domain level Method (DM), and Combined Method (CM).

Application level Method (AM)

The Application level test case selection method, consists of selecting test cases in order to thoroughly test products one after another. In this chapter, this method is applied to **maximize the structural coverage of each individual product** (i.e., to maximize the PSC measure described in Section 6.3.3). To this end, we first prioritize products of the product suite as it has been shown that in time-budget constrained scenarios it improves fault detection results [PSS+16]. Next, products are sequentially tested, selecting test cases according to their structural coverage until 100% of the PSC is obtained. This process is continued until the last product of the product suite is tested, or the Time Budget (TB) is consumed. The overall process for this method is described in Algorithm 2.

Note that the pseudo-function $get\_UnusedTC\_BestPSC$ provides each test case with the highest product structural coverage increment (i.e., highest PSC coverage increment) among all unused test cases of the selected product ($curProduct$), in the same way as the additional greedy algorithm [YH07]. If two test cases with equal execution time provide identical coverage increment, the pseudo-function selects the first one.

Table 6.2 presents the application of this method considering the example in Table 6.1, for a given time budget of 20 seconds. The algorithm selects test cases from the firstly prioritized product P_3 according to their test requirements coverage. Thus, the first test case to be selected is TC_{2P_3} as it covers 10 test

Algorithm 2: Application level method

```
1 prioritize products;
2 curProduct = select highest priority product;
3 while ( curProduct <= lastProduct ) & ( consumedTB < TB ) do
4     while ( PSCoverage < 100% ) & ( consumedTB < TB ) do
5         nextTC = get_UnusedTC_BestPSC( curProduct, PSCoverage
           );
6         solution = solution ∪ nextTC ;
7         PSCoverage = updatePSCoverage( solution );
8         consumedTB = updateTimeBudget( solution );
9     curProduct = select next highest priority product;
```

Table 6.2: Test case selection example for Application level method for a given time budget of 20 seconds.

Application level Method			
Product	nextTC	PSCoverage	consumedTB
P3	TC_{2P_3}	77%	6
P3	TC_{1P_3}	100%	10
P2	TC_{3P_2}	67%	11
P2	TC_{2P_2}	100%	19

requirements. The next test case in product P_3 with the highest increase of test requirement is TC_{1P_3}, incrementing the covered test requirements by 3. Since the remaining test case does not increase the structural coverage of P_3 and the remaining time budget is not consumed (i.e., there are 10 seconds left), the next prioritized product is selected (i.e., P_2). Similarly to the previous product, two test cases are selected; firstly TC_{3P_2} which increments the test requirements coverage by 8 and secondly, TC_{2P_2} which increments by 4. The consumed time budget (i.e., consumedTB) for executing these four test cases is 19 seconds. Since the structural product coverage of the first two prioritized products is fully achieved (i.e., PSC of P_3 and P_2 is 100%), the next prioritized product is selected (i.e., P_1). However, as test cases in P_1 have a higher TET than 1 second, it is not possible to select further test cases because the time budget would be exceeded. Thus, the AM test case selection method would return test cases $TC_{2P_3}, TC_{3P_3}, TC_{2P_2}$ and TC_{2P_2}.

Domain level Method (DM)

The Domain level test selection method consists of selecting test cases considering **the structural coverage of the entire product line** (i.e., maximizing the PLSC measure described in Section 6.3.3). To this end, as any test case from the derived products of the product line can be selected without product prioritization restriction, the test case that increments the most PLSC is selected each time. Algorithm 3 synthesizes the DM test selection method.

Algorithm 3: Domain level method

```
1 while ( nextTC <= lastTC ) & ( consumedTB < TB ) do
2     nextTC = get_UnusedTC_BestPLSC( PLTS, PLSCoverage );
3     solution = solution ∪ nextTC ;
4     consumedTB = updateTimeBudget( solution );
5     PLSCoverage = updatePLSCoverage( solution );
6     if ( PLSCoverage = 100% ) & ( consumedTB < TB ) then
7         PLSCoverage = 0;
```

Note that $PLTS$ stands for the Product-Line Test Suite, which comprises all test cases created for the product line. It is also worth mentioning that the pseudo-function *get_UnusedTC_BestPLSC* provides the test case each time with the highest test requirements coverage increment for $PLTS$ among all unused test cases. Once full PLSC is obtained, if the time budget has not been consumed, the PLSC measure is reset to 0, and the process is re-started by selecting test cases that have not been previously selected.

Table 6.3: Test case selection example for Domain level method for a given time budget of 20 seconds.

Domain level Method			
Product	nextTC	PLSCoverage	consumedTB
P3	TC_{2P_3}	63%	6
P2	TC_{2P_2}	94%	14
P1	TC_{1P_1}	100%	16
P2	TC_{3P_2}	50%	17
P3	TC_{3P_3}	56%	19

Table 6.3 presents how the Domain level method is applied considering the example in Table 6.1, for a given time budget of 20 seconds. The algorithm can select from any of the 9 test cases that compose the $PLTS$ to maximize

the structural coverage of the entire product-line (i.e., PLSC). The first test case to be selected is TC_{2P_3}, as it is the test case that provides the highest test requirements coverage increment (i.e., 10 new test requirements are covered). The next selected test cases are TC_{2P_2} and TC_{1P_1} as they provide the highest test requirements increment in terms of the PLSC (i.e., TC_{2P_2} increments the test requirements coverage by 5 and TC_{1P_1} by 1). After selecting these three test cases, 100% of the PLSC is achieved (i.e., the 16 test requirements are covered) while the time budged is not consumed (i.e., 16 seconds are employed). Since there is time left (i.e., 4 seconds left), the remaining TB is used to complete it with other TCs that could potentially detect new faults. Therefore, PLSC coverage is set to 0 and the next test case is selected among those test cases that have not been selected before and provide the highest PLSC coverage. The next selected test case is TC_{3P_2} as it provides the highest increment of test requirements (i.e., 8 test requirements are covered) among unused test cases, and consumes 1 extra second time budget. Finally, although TC_{1P_3} increments more test requirements, as there is not enough time budget left to accommodate it, the TC_{3P_3} test case is selected, which increments the covered test requirements by 1 and consumes 2 seconds. For the provided example, DM test case selection method would return test cases TC_{2P_3} , TC_{2P_2} , TC_{1P_1}, TC_{3P_2} and TC_{3P_3}.

Combined Method (CM)

In this test selection method, we combine both of the previous test selection methods. As in the Application level method, first, the products are prioritized. However, instead of selecting test cases considering the product coverage, the product line coverage is considered (i.e., the PLSC coverage). The main difference is that the **test cases are selected taking into account the prioritization of the products but considering the coverage of the entire product line**. Algorithm 4 synthesizes the CM.

It is worth mentioning that in this method, while there are test cases that provide PLSC increment, the selected current product ($curProduct$) is maintained. However, when none of the remaining test cases corresponding to the current product increments the PLSC, the next prioritized product is chosen. Similarly to the domain level method, when 100% of coverage is reached and the time budget has not been consumed, the PLSC measure is reset (i.e., PLSC = 0) and the test case selection is continued.

Algorithm 4: Combined method

```
1  prioritize products;
2  curProduct = select highest priority product;
3  while ( curProduct <= lastProduct ) & ( consumedTB < TB ) do
4      while ( ΔTR_nextTC > 0 ) do
5          nextTC = get_UnusedTC_BestPLSC( curProduct,
             PLSCoverage );
6          solution = solution ∪ nextTC ;
7          consumedTB = updateTimeBudget( solution );
8          PLSCoverage = updatePLSCoverage( solution );
9          if ( PLSCoverage = 100% ) & ( consumedTB < TB ) then
10             PLSCoverage = 0;
11     curProduct = select next highest priority product;
```

Table 6.4: Test case selection example for combined method for a given time budget of 20 seconds.

Combined Method			
Product	nextTC	PLSCoverage	consumedTB
P3	TC_{2P_3}	63%	6
P3	TC_{1P_3}	81%	10
P2	TC_{2P_2}	94%	18
P1	TC_{1P_1}	100%	20

Table 6.4 presents how the combined method is applied considering the example in Table 6.1, for a given time budget of 20 seconds. Similarly to the Application level method, this algorithm selects test cases from the firstly prioritized product P_3. However, instead of measuring the capability of the test case to increase the coverage of the product (i.e., PSC), the product-line structural coverage (i.e., PLSC) is measured to select test cases. Thus, the first test case to be selected is TC_{2P_3}, as it provides the highest test requirement coverage increment to the product line (i.e., an increment of 10 test requirements) among all those test cases available for product P_3. The next test case TC_{1P_3} provides a 3 test requirement increment to the product line. Since there are no available test cases capable of incrementing the PLSC, the next product is selected (i.e., P_2). Among P_2 product test cases, only TC_{2P_2} provides an increment of test requirements (i.e., 2 test requirements are covered), which is included in the test case selection, and the next product is

selected. Between the P_1 product test cases, to complete the PLSC and respect the time budget, test case TC_{1P_1} is selected. This test selection method would return test cases $TC_{2P_3}, TC_{1P_3}, TC_{2P_2}$ and TC_{1P_1}.

6.4 Evaluation

The general purpose of this experiment was the comparison between three test case selection methods when using different structural coverage criteria in a time-constrained scenario. To this end, the following three Research Questions (RQs) were defined:

- **RQ1 - Sanity Check:** *Are the proposed test case selection methods effective when compared to the baseline?* The first RQ examines whether the proposed test case selection methods, described in Section 6.3.4, are effective when compared with the baseline. The defined baseline consists of applying all existing test cases from the test suite for each prioritized product while there is time budget. This research question was defined as a sanity check to verify that the test case selection problem was non-trivial to solve. To tackle this research question, the fault detection effectiveness of the proposed three test case selection methods were compared with the baseline.
- **RQ2 - Methods Evaluation:** *Which of the proposed test case selection methods fares best in terms of fault detection capability?* The second RQ was defined to identify which test case selection method performs best when detecting faults in a time-constrained scenario. To address this RQ the fault detection effectiveness of the three proposed test case selection methods was compared.
- **RQ3 - Coverage criteria Evaluation:** *Which of the selected coverage criterion fares best in terms of fault detection capability?* The third RQ is defined to identify which coverage criterion performs best when detecting faults in a time-constrained scenario. To address this RQ the fault detection effectiveness of the analyzed coverage criteria were compared.

6.4.1 Experiment design

The following sub-sections describe how the experiment was designed to evaluate the defined RQs.

Case study

We selected the CarWindow (CW) case study (further detailed in Section 4.4.2 for the experiment. To describe the variability of this cyber-physical system product line, it was employed a feature model composed of 30 different features. Therefore, more than 11.000 products could be configured with this product line. In order to reduce the number of product configurations to be tested, we derived the product suite using the pairwise coverage criterion and the ICPL algorithm [JHF12], obtaining a total of 28 product configurations. The simplest product configuration was composed of 4 features, whereas the most complex product configurations were composed of 19 features. On average, product configurations were composed of 12 features.

Based on the feature model of the case study, a 150% model was designed using MATLAB/Simulink, containing 227 Simulink blocks structured on 4 depth levels. A 150% model contains the union of all blocks required to model any variant of the SPL. In a negative variability approach, blocks related to deselected features are removed from the 150% model to derive particular product configurations, also known as 100% models. For each derived product configuration, a specific MATLAB/Simulink 100% product model was obtained (hereinafter referred to as product). The resulting product suite, composed of the 28 products, was prioritized according to the Variability Coverage and Cyclomatic Complexity (VC&CC) prioritization criterion, based on the good performance this product prioritization criterion had shown in the past [SSRC14a].

Test suite generation

In addition, for each of the derived products, the test cases were automatically generated employing a white-box approach. In our context, a test case consists of a set of signals that stimulate the system's input. Simulink Design Verifier (SLDV) was employed to generate test cases following the MC/DC criterion.[3]

[3]When MATLAB/Simulink is configured to generate test cases with the MC/DC criterion, an aggregated test suite to satisfy the following three coverage criteria is generated: Condition Coverage, Decision Coverage, and Modified Condition/Decision Coverage criteria. For further details visit the following link https://es.mathworks.com/help/sldv/ug/model-coverage-objectives-for-test-generation.html.

On average 42 test cases were generated per product. Thus, in total, 1,184 test cases were generated for the 28 products.

Evaluation metric

We employed mutation testing to evaluate the effectiveness of the experiment, as it has been found to be a good substitute of real faults [JJI+14]. Concretely, we employed the **Mutation Score** (MS) metric to assess the fault detection capability of the proposed methods. MS measures the percentage of mutants detected by a specific test suite.

In mutation testing, a copy of the original system (named mutant) is created, where a fault (also known as mutation) is injected. When both the original system and the mutant are executed, if obtained results are different, it is considered that the mutant has been "killed", therefore a fault is detected. Since our case study was modeled in MATLAB/Simulink, mutants were created employing the mutation operators proposed by Hanh et al. [HBT16]. Each generated mutant consisted of a copy of the original system, where the injected fault was related to a specific feature associated to the 150% model. This mutant was later selected when a specific product included the faulty feature into the configuration. We generated a total of 200 mutants for the 150% model. Although it is not a large number of mutants compared to other approaches, it is noteworthy that MATLAB/Simulink models are heavy to execute. Note however, that our approach employs similar or more mutants to other approaches employing MATLAB/Simulink [AWSE16b, AWA+18, MASE17, MNBB16].

In our case study, each specific product (i.e., the 100% product model) was generated including one copy of the non-mutated system and a specific number of mutants. The final number of mutants per product was determined by the corresponding feature selection, where unrelated mutants were removed (i.e., mutants related to features not included in the product were removed). When test cases were executed in each product configuration model, all copies of the system (i.e., non-mutated and mutated copies) were simulated with the same inputs (i.e., previously generated test cases), and their outputs were compared. When certain mutant output differed from non-mutated output, the mutant was killed, detecting a fault.

The results obtained from simulating the selected product models (which included mutants) with generated test cases were post-processed to get the mutants detection data for each specific test case. According to the gathered mutants detection results, to obtain the final set of mutants we first removed

undetectable mutants (i.e., those mutants that were not detected by any of the test cases) with the aim of removing equivalent mutants. Secondly, we removed duplicated mutants (i.e., mutants equivalent to one another but not to the original program), as Papadakis et al. recommended in [PJHLT15]. Lastly, we removed those mutants that were killable by all test cases. Subsequently, the remaining number of mutants for the experiment was reduced from 200 to 129.

The test cases generated with MATLAB/Simulink were designed to satisfy a set of test requirements[4]. After simulating products with test cases, the coverage data was gathered. On average 99% coverage rate was obtained, which means that 99% of the defined test requirements were satisfied in terms of coverage during simulation. However, MATLAB/Simulink was not capable to satisfy the remaining test requirements with generated test cases. When 100% of coverage is reported in the following sections, it must be taken into account that full coverage of the reachable coverage rate (i.e., 99%) was obtained. Moreover, for each of the generated test cases, a specific number of test requirements were related, according to the coverage data generated during simulation.

Artificial problems

We selected 25 different time budget samples as artificial problems for the experiment. The first 10 time budget samples were defined from 25 seconds to 250 seconds with increments of 25 seconds between each artificial problem. The remaining 15 time budget samples were defined from 250 seconds to 4,000 seconds, with increments of 250 seconds. The time budget samples were selected based on preliminary experiments.

6.4.2 Results

RQ1 Results

To answer RQ1, the mutation score of the three proposed test case selection methods were compared with the baseline. Figures 6.3a, 6.3b and 6.3c depict [5] this comparison for the specific DC, CC and MC/DC structural coverage

[4] In the particular case of MATLAB/Simulink, the term Test Objective is the equivalent designation employed for the test requirements described in this work (refer to section 3.3 for further details).

[5] Results after 1000 seconds time budget present a convergence of the compared series that tend to the 100% mutation score (both of the test selection methods as well as of the coverage criteria). In order to improve the visualization of the most relevant information obtained, the graphs have been limited to time budgets between 25 and 1000 seconds.

criteria respectively. Furthermore, we report the number of artificial problems for which the proposed methods performed better than the baseline in Table 6.5. For each of the coverage criteria (i.e., DC, CC and MC/DC) 3 columns are reported. The column A > B presents the number of artificial problems for which the method in column A showed better MS performance than the method B. B > A represents the opposite. The column A = B reports the number of artificial problems for which there was no difference between methods. As it can be seen in the aforementioned figures, Domain level methods outperformed the baseline in all cases. The Application level and combined level methods, overall, outperformed also the baseline. However, in a reduced number of cases the baseline was better than application or combined level methods due to the product prioritization constraint. Specifically, when Table 6.5 is observed for the DC criterion, the Application level method obtained higher MS than the baseline in 17 out of 25 artificial problems. The combined method performed better in 18 of the 25 artificial problems. Lastly, the Domain level method performed better than the baseline in 21 of the 25 artificial problems. Similar results for remaining coverage criteria were obtained.

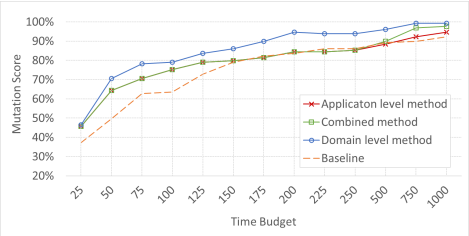
Table 6.5: Summary of test case selection methods compared with the baseline (BL) for each coverage criterion.

Methods		DC			CC			MC/DC		
A	B	A >B	A = B	B >A	A >B	A = B	B >A	A >B	A = B	B >A
AM	BL	17	4	4	12	10	3	13	7	5
DM	BL	21	4	0	21	0	4	21	4	0
CM	BL	18	4	3	16	6	3	18	6	4

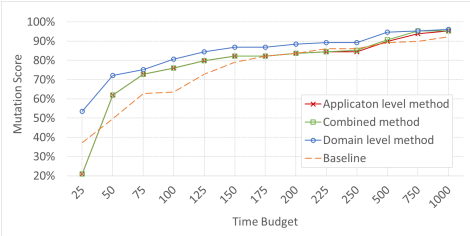
Table 6.6: Summary of coverage criteria comparison for each test case selection method.

Criteria		AM			DM			CM		
A	B	A >B	A = B	B >A	A >B	A = B	B >A	A >B	A = B	B >A
CC	DC	8	8	9	5	0	20	6	9	10
DC	MC/DC	7	14	4	9	11	5	6	15	4
CC	MC/DC	6	11	8	4	0	21	5	10	10

(a) Mutation Score of compared test case selection methods for CD coverage criterion



(b) Mutation Score of compared test case selection methods for CC coverage criterion



(c) Mutation Score of compared test case selection methods for MC/CD coverage criterion

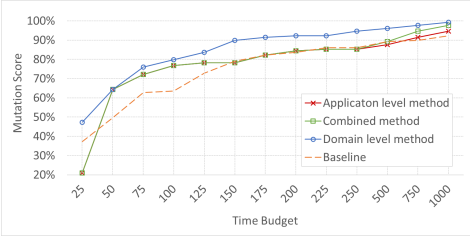


Figure 6.3: Mutation score obtained by the Application level, Domain level and combined methods, compared with the baseline, across time budgets from 25 to 1000 seconds, corresponding to the specific DC (6.3a), CC (6.3b) and MC/DC (6.3c) coverage criteria.

RQ2 Results

Figures 6.3a, 6.3b and 6.3c depict the results for the different test case selection methods proposed in Section 6.3.4. In addition, we report the number of artificial problems for which a method was better than the other in Table 6.7. The structure of Table 6.7 is identical to the structure of Table 6.5. Specifically, Table 6.7 compares the Application level, Domain level and combined test case selection methods among them.

When considering the figures, figure 6.3a shows the results of the three methods proposed in Section 6.3.4 for the DC criterion, figure 6.3b for the CC criterion and figure 6.3c for the MC/DC criterion. Overall for the lower time budgets, the Domain level method outperformed the remaining proposed methods as shown in the three figures. In fact, as Table 6.7 shows, when the DC coverage criterion was used, the Domain level method obtained higher MS than the Application level method in 17 out of 25 artificial problems, and in 16 out of 25 artificial problems for the combined method. In the case of CC

Table 6.7: Summary of test case selection method comparison for each coverage criterion.

Methods		DC			CC			MC/DC		
A	B	A >B	A = B	B >A	A >B	A = B	B >A	A >B	A = B	B >A
AM	DM	0	8	17	6	0	19	0	8	17
AM	CM	0	18	7	0	16	9	0	17	8
DM	CM	16	9	0	14	3	8	13	12	0

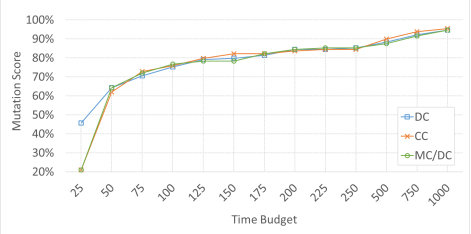
coverage criterion, the Domain level method outperformed the Application level one in 19 out of 25 artificial problems and the combined method in 14 out of 25 artificial problems. Lastly, for the MC/DC criterion, the Domain level method outperformed the Application level method in 17 out of 25 artificial problems and the combined method in 13 out of 25 cases. The Domain level method performed better than the remaining ones when the time budgets were low as it can be seen in Figures 6.3a, 6.3b and 6.3c. When these time budgets were increased, the differences in MS between both, the application as well as the combined methods with respect to the Domain level method decreased.

RQ3 Results

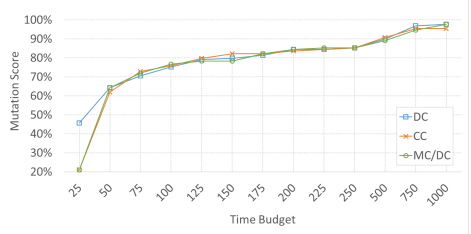
Figures 6.4a, 6.4b and 6.4c depict the results for the compared different coverage criteria. In addition, we report the number of artificial problems for which a coverage criterion was better than the other in Table 6.6. Specifically, Table 6.6 compares DC, CC and MC/DC coverage criteria among them.

As far as the figures are concerned, figure 6.4a shows the results of the three coverage criteria for the Application level test case selection method, figure 6.4b for the combined method and figure 6.4c for the Domain level method. Results of the three graphs show that the three coverage criteria get a very similar mutation score throughout the artificial problems of the experiment for most of the cases. More concretely, as it is presented in Table 6.6, when the Application level method was applied, comparing the DC criterion with the MC/DC criterion, column $A{=}B$ reported the highest value of the comparison (i.e., in 14 out of the 25 artificial problems DC and MC/DC obtained the same MS value). When the Domain level method was applied, both the DC and MC/DC criteria performed better than the CC criterion, i.e., in 20 out of 25 artificial problems the DC criterion obtained better MS; while the MC/DC criterion obtained better MS in 21 out of 25. This is also shown in Figure 6.4c,

(a) Mutation Score of compared coverage criteria for Application level method



(b) Mutation Score of compared coverage criteria for combined method



(c) Mutation Score of compared coverage criteria for Domain level method

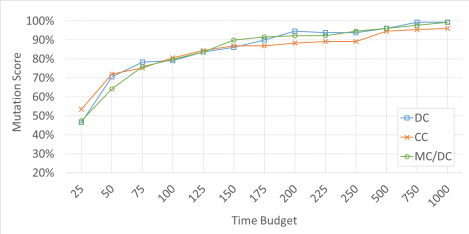


Figure 6.4: Mutation score obtained by DC, CC and MD/DC criteria across time budgets from 25 to 1000 seconds, corresponding to the specific Application level (6.4a), combined (6.4b) and Domain level (6.4c) test case selection methods.

where the CC criterion in most cases scores lower than the DC and MC/DC criteria. However, the difference between the criteria is not substantial. When the combined method was applied, similarly to the application level method, comparing the DC criterion with the MC/DC criterion, column $A{=}B$ reported the highest value of the comparison (i.e., in 15 out of the 25 artificial problems DC and MC/DC obtained the same MS value).

6.4.3 Discussion

In this section, we present the analysis and discussion of the results obtained for each of the RQs.

RQ1 - Sanity Check:

The first RQ aimed at examining whether the proposed three test case selection methods were effective when compared with the baseline. The results of the

experiment showed that the mutation score obtained by the proposed test case selection methods outperformed the baseline in most of the cases.

> *The problem to be solved is not trivial and that the proposed methods for the test case selection provide an improvement in fault detection capability.*

RQ2 - Methods Evaluation:

RQ2 aimed at identifying which of the proposed test case selection methods fared best when detecting faults in a time-constrained scenario.

> *The obtained results suggest that at lower time budgets, the Domain level method performed better than the remaining two methods.*

A possible reason for this could be that the Domain level method can select all the 1184 test cases and increase the product line coverage, whereas the other two methods are restricted by the product prioritization and can select only test cases of a specific product at a time. This permits the Domain level method to select better test cases at any time. When the time budgets were higher, the mutation score of the three test case selection methods reached the highest score. The reason for this could be that a higher time budget allows for selecting more test cases, and thus, the chance for detecting all the mutants increases.

RQ3 - Coverage criteria Evaluation:

RQ3 was defined to assess the effectiveness of the analyzed coverage criteria.

> *The results suggested that overall, all the coverage criteria performed similarly considering the mutation score.*

It is well known that the MC/DC coverage is a stronger criteria and thus should find more faults than the remaining ones. A possible reason for the MC/DC having similar results as the CC and DC could be that, for getting higher MC/DC coverage, longer test cases in terms of TET are required. As our methods limit the selection of test cases depending on their time budget, this fact could have reduced the possibility of having higher mutation scores when employing the MC/DC coverage criterion.

6.4.4 Threats to validity

This section identifies the threats that could invalidate the performed empirical evaluation. An **internal validity** threat that we identified is related to the number of employed mutants; we first used 200 mutants and when removing the duplicated, equivalent and trivial mutants, only 129 mutants were employed, which might not be a high number. However, we mentioned the high cost that executing mutants within MATLAB/Simulink models involves. Note that in other experiments where MATLAB/Simulink models are used, a lower number of mutants is employed (e.g., 13 to 44 mutants [MNBB16], or 9 to 96 [AWA+18]). Furthermore, we reduced this threat by removing trivial, equivalent and duplicated mutants and by introducing different mutation operators across different parts of the system. An **external validity** threat refers to the number of case studies used. As the experiment has employed only one case study, results should not be generalized. However, we reduced this threat by employing a real-world application case study of medium size. In addition, we divided the evaluation into 25 different artificial problems.

6.5 Related Work

Testing SPLs has garnered attention in the last few years. According to literature reviews and surveys [ER11, LHFRE15], most testing SPLs works focus on deriving a set of relevant products from a variability model [EBG12, POS+12, CDS08, HPHT15], typically a feature model, and following a set of different approaches, such as combinatorial (e.g., pairwise) [LHFRE15]. Other approaches focus on prioritizing these derived products with different objectives, typically increasing the fault detection rate (e.g., [SSRC14a, AHTM+14, AHTL+16, DPC+15, PSS+16]), but also others, such as reducing the switching costs [WNK14]. Both approaches, sampling and prioritizing products focus on the domain engineering level. Conversely, the Domain level method that we propose considers both test levels (i.e., it selects products and test cases with the aim of maximizing the structural coverage as much as possible, as detailed in Section 6.3.4).

The approach proposed by Devroey et al. [DPC+14] use behavioral models when testing product lines, such as the featured transition systems. In this work the behavioral coverage driven testing is stated and coverage criteria are redefined at the domain engineering level.

Other approaches focus on the application engineering level by selecting a

relevant subset of test cases to test a specific product (e.g., [WAGL16a, WAG15, AWSE16a, LFN+17]) or, once these tests have been selected, prioritizing them [WBA+14, AWSE16b, LLL+15]. Most of these application engineering level approaches consider the historical data of test cases (e.g., the fault detection capability of test cases) to guide the test case selection. Taking historical data related to failure into account is interesting and makes the test selection or prioritization effective, as demonstrated in several studies [YH07, AWSE16b, AWSE16a]. However, to obtain historical data, it is necessary to execute the test cases several times, something that is not always feasible. Our work differs from these approaches because the methods we propose consider structural coverage information to guide the selection of test cases instead of employing historical data of test cases. Other application engineering level approaches such as Stricker et al. [SMP10] conduct the test case selection for SPLs employing structural coverage information as we do. However, this approach relies on data-flow dependencies to select test cases whereas our work is centered on control-flow coverage criteria. Another difference is that our work evaluates results using the mutation testing technique.

6.6 Conclusions

In this chapter we propose different test case selection methods based on structural coverage. To this end, we employ two metrics for structural coverage and propose three different test case selection methods based on these metrics. In addition, we conducted this study for three different coverage criteria, i.e., DC, CC, MC/DC. An empirical evaluation was carried out with a case study from the automotive sector applying the mutation testing technique. Results suggest that the Domain level method, which focuses on selecting test cases considering the overall coverage of the SPL, obtains better results than the other methods for the most reduced time budgets.

On the basis of the results obtained, it has been possible to validate the optimization goal of HCCPSs testing by considering the structural coverage information. With this validation the second line of the exploratory work was concluded.

Chapter 7

Dynamic prioritization of Products and Test Cases for testing HCCPS

Contents

In this chapter we provide the details of the main contribution. Section 7.1 contextualizes the presented contribution in the theoretical framework. Section 7.2 provides technical motivation for this contribution. Section 7.3 presents our approach for dynamic test prioritization of product lines. The adaption of our approach to the specific context of configurable simulation models is presented in Section 7.4. The empirical evaluation of our approach is described in Section 7.5, including the experimental setup, results, analysis and discussion. Section 7.6 describes related work relevant to test prioritization. Finally, Section 7.7 summarizes the work and presents conclusions and future work.

7.1 Contribution overview

The research work presented in this chapter details the main contribution of the thesis. Figure 7.1 highlights this work (in green) in the context of the entire theoretical framework (further details of the theoretical framework are provided in Section 4.3).

In the exploratory stage, two works and one additional input were produced. On the one hand, the **first exploratory work** analyzed the proposal for the selection and prioritization of small groups of test cases in an iterative way (further described in Chapter 5). On the other hand, the **second exploratory work** analyzed the possibility of using information from both domain and Application levels of the product line (further described in Chapter 6).

Additionally, as a **final input** to the exploratory stage, a study was carried out to determine quality metrics for testing configurable simulation models. In this work we describe 5 groups of metrics categorized into white-box and black-box testing techniques. The study collects existing metrics (some of them proposed by third-party authors and others proposed within the framework of this thesis) and adapts them to the context of configurable simulation models. Among the studied metrics we highlight two, due to their relation to this main contribution: (1) the metric for measuring similarity between test cases and (2) the metric for measuring similarity between configurations (products) and the test cases. Refer to Section 2.4.3 for further details.

From Chapter 5 we conclude that an (i) iterative approach in combination with a strategy to select incrementally a small number of test cases each time is effective; and (ii) similarity metrics assist in fault detection. From Chapter 6 we conclude that (i) metrics combining both product-level and product-line-level information are effective; and (ii) combining the selection of products and test

cases improves effectiveness when detecting faults (despite modifying the order established in the product prioritization.) In addition to the input related to quality metrics, we highlight similarity-based metrics that provide an effective tool in the context of HCCPS.

As a result of analyzing the conclusions obtained from the exploratory stage, the basis for the next work were established: (1) it is necessary to extend the research on the selection and prioritization of both products and test cases jointly. (2) It is necessary to extend the research on testing small test sets in an iterative way. (3) It is necessary to extend the investigation of the use of information from both domain and Application levels when optimizing HCCPS testing. (4) There is a need to extend research into the use of similarity-based quality metrics for testing configurable simulation models.

Based on the results and the subsequent conclusions drawn from the exploratory stage, the **hypothesis** for the main contribution was formulated, which consists of using the results of the previously executed test cases to perform a dynamic prioritization of the test cases and products to improve fault detection capability of HCCPSs.

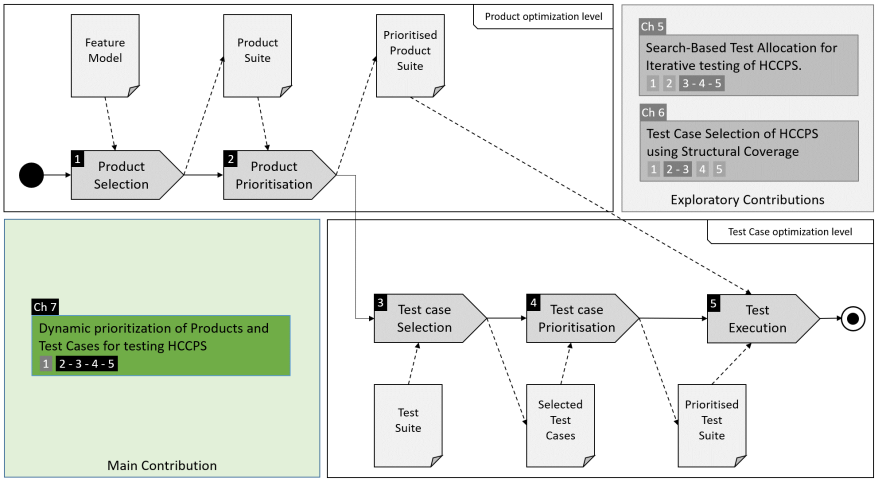


Figure 7.1: Method proposed as main contribution in the context of the theoretical framework. The method "Dynamic prioritization of Products and Test Cases for testing HCCPSs" (green colored), which involves steps number 2, 3, 4 and 5.

The proposed approach develops the *core concept* by combining 4 out of the 5 steps of the traditional HCCPS testing process (i.e., steps 2 to 5 of Figure 7.1). Furthermore, the approach proposes a technique to select and prioritize test cases and products simultaneously. More specifically, we propose

a dynamic test prioritization approach in order to optimize the testing process of SPLs by increasing the fault detection rate. In contrast to traditional static test prioritization, our dynamic test prioritization leverages information of test cases being executed in specific products. Processing this information, the initially prioritized tests are rearranged in order to find non-discovered faults. The proposed approach is valid for any kind of SPL, but we have adapted it to the context of HCCPS simulation-models, an area where testing is especially time-consuming and optimization methods are paramount.

The approach was empirically evaluated by employing the CW and IT case studies (further detailed in Section 4.4). The results of this evaluation reveal that the proposed test prioritization approach improves both the static prioritization algorithm and the selected baseline technique (Refer to Section 7.3.2 for further details of the static prioritization algorithm).

Consequently, this research work allowed us to empirically validate the main contribution of dynamic prioritization of products and test cases for testing HCCPSs.

7.2 Introduction

Product line systems testing is a time-consuming activity [LHFRE15, PSK+10]. This is usually because the number of potential product configurations to be tested can be huge. Several approaches have been proposed to make SPL system testing feasible, some focusing on reducing and prioritizing the number of configurations to test, others on selecting and prioritizing tests to apply.

Test prioritization aims at increasing the fault detection rate by ranking test cases according to their likelihood of finding faults. This practice has shown to optimize the testing process of several industrial systems from different domains [BX16, DNABL13, ERL11, EYHB15, MGS13, SNS+18, SGMM17, THHB14].

Since this activity was successful, the product line engineering community adapted test prioritization to the context of configurable systems. Most of test prioritization approaches in product line engineering focused on the domain engineering level, aiming at prioritizing the order at which products were tested [AHTL+16, BLLS14, DPC+15, HPP+14, JHF12, SSRC14a]. Other approaches focused on the application engineering level, aiming at prioritizing the order of test cases based on several criteria (e.g., fault detection capability, test execution time, etc.) [AWSE16a, AWSE19, EBA+11, LLL+15, WBA+14].

A drawback of the former was that once the products were prioritized, the

time it took to execute tests in each system variant was not considered. It was assumed that if a system variant contained one or more errors, these were detected. However, testing each variant is a non-trivial process (test cases must achieve certain degree of test coverage, different functional parts have to be tested, etc.). Even if this was considered, in practice, it could happen that the available time budget for testing all the selected configurations may be consumed before task completion. For the latter, only single system variants were considered. However, once the tests were ranked, its execution was fixed, without possibility to change of system variants at certain point. Furthermore, for some cases [HPH12, KAF12, KIJT18, KP02, LCTK13, MGS13, NWF+15, SCC16], a historical database was required to rank test cases based on some adequacy criteria. While this has shown to be effective, this database is not always available, or else it requires a long start-up phase to be effective.

In this approach we prioritize test cases and products at the same time (i.e., we give the chance of executing a test case in a specific product first, and later executing another test case in another product, to later come back to the first product). As an input to our approach, we consider a set of prioritized products and test cases. Later, dynamically, we re-organize the order at which these are tested, by considering the verdicts of tests. This dynamic test prioritization approach is general to any kind of product line engineering testing method, but it requires adaption depending on the domain. In this chapter we present how we have adapted it for testing configurable simulation models, which are typically used to test Cyber-Physical System (CPS) product lines [AWM+19, AWSE19, AIH15]; this choice was taken because testing these systems in practice is extremely time-consuming and effective test optimization techniques are required [BNSB16, LNLB19, MNBB19].

7.3 Dynamic test prioritization of product lines

This section presents an approach for the dynamic test prioritization of product lines. The approach focuses on the dynamic selection of the next best pair (composed of a test case and a product) to be tested. For this purpose, a three phase process is conducted: in the first phase, all valid pairs of products and test cases are build. The second phase statically prioritizes valid pairs based on test similarity metrics (refer to Section 7.3.1 for further details). Finally, during the third phase, statically prioritized pairs are dynamically reallocated and executed, considering that pairs containing products with higher Fault

Detection Capability (FDC) are more likely to reveal other faults. At the end of the process, dynamically prioritized pairs are obtained. An overview of the approach is presented in Figure 7.2 and further details of each phase are provided in the rest of the section.

Let $PS = \{P_1, P_2, \cdots, P_{np}\}$ be a product suite with np number of products derived from the PL product line. And let $TS = \{TC_1, TC_2, \cdots, TC_{nt}\}$ be a test suite with nt number of test cases generated to test products of the PL product line.[1]

Definition 11: Valid Pair

A valid pair PR_i consists of a couple formed by a specific product P_j, such that $P_j \in PS$, and a test case TC_k, where $TC_k \in TS$, if it is guaranteed that the test case TC_k can be executed on the product P_j.

Suppose for example that TC_1 (being $TC_1 \in TS$) can be executed with product P_2 (being $P_2 \in PS$), hence they form the valid pair $PR_1 = (TC_1, P_2)$.

In software product lines not all test cases can usually be executed with all products, so the final number of test cases available to build pairs is reduced to nt', where $nt' \leq nt$. In addition, not all products necessarily have test cases to run them, therefore the final number of products available to build pairs is reduced to np', where $np' \leq np$.[2]

The first phase of the approach consists of one step (i.e., *Step 1: Build Valid Pairs* in Figure 7.2) where all valid pairs are built. To this end, a set of products (*Product Suite* in Figure 7.2) derived from the product line and a set of test cases (*Test Suite* in Figure 7.2) to test the product line are provided as inputs. During this phase, only valid pairs are conformed and saved into the *Set of valid Pairs* data structure.

The second phase of the process addresses the static prioritization of pairs in one step (i.e., *Step 2: Prioritize Statically* in Figure 7.2). In this step, the *Set of valid Pairs* data structure is received as input and the Static Pair Prioritization algorithm is applied based on a similarity criterion. Note that any state-of-the-art prioritization criterion [DPC+15, EBA+11, WBA+14] can

[1] Note that PS and TS consist of a generalization of the definitions proposed in Section 4.4.1 of $SMPS$ and $TS_{150}\%$ respectively, which are redefined in the generic product line engineering context of the approach described in this section.

[2] In our study, all generated test cases can be executed with all derived products. Refer to Section 7.5.2 for details of the product and test case generation.

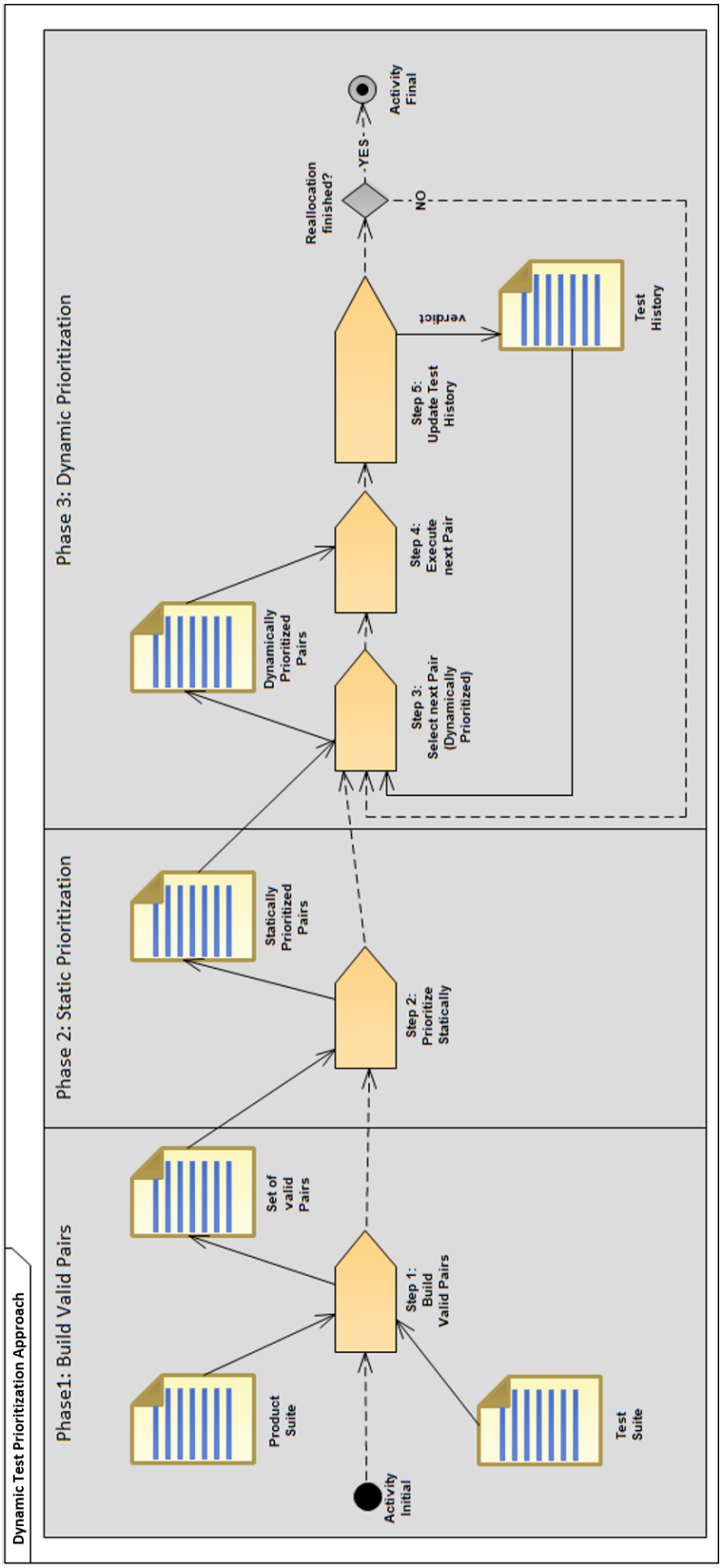


Figure 7.2: Dynamic Test Prioritization Approach Overview

be employed to statically prioritize valid pairs.[3] As a result of the second phase, the set of *Statically Prioritized Pairs* is obtained.

During the third phase, which is composed of steps 3, 4 and 5, the dynamic pair prioritization is carried out. In the third step (i.e., *Step 3: Select next Pair (Dynamically Prioritized)* in Figure 7.2), the next best pair to be executed is selected based on the existing test history. To this end, the pair that contains the product with highest fault detection capability is selected. In the fourth step (i.e., *Step 4: Execute next Pair* in Figure 7.2), the selected pair is executed, so that in the fifth step (i.e., *Step 5: Update Test History* in Figure 7.2), the Test History is updated. The third, fourth and fifth steps are repeated iteratively until all statically prioritized pairs are reordered in the resulting *Dynamically Prioritized Pairs* output.

Note that every time a pair is executed (Step 4), the verdict is saved into the *Test History*. To this end, we have implemented a prototype of the approach in MATLAB.[4] In the prototype, we perform the maintenance of the Test History in table format indicating for each of the executed pairs (i.e., product column, test-case column) whether its execution has been successful or not (i.e., pass, fail, etc.) together with the pertinent metadata. This information is used to calculate the fault detection capability of the product dividing the number of faults found in the product by the number of executions. We recall that in this sense, we make no difference between faults (i.e., if a fault is exhibited with more than one test case, the product is considered to have failed in all cases). This is because we assume that detecting one fault does not necessarily lead to its localization, which is beyond the scope of this paper.

The reason why we re-prioritize at the product level is because we hypothesize that a product where we have detected a fault has a higher probability of having more faults than a product where we did not find faults. This could be due to different reasons. For instance, one such reason is that the product where we have found a fault has more complex features. In addition, it has been found in the past that a feature, or the interaction between more than one feature might have more than one fault [SSPRC15]. Another reason is that focusing on buggy areas has been found to be effective to uncover severe bugs in complex systems like the Google code-base [WAC12].

[3]For this study, we have employed a similarity based criterion to prioritize pairs statically, based on weighted measures that combines both product similarity and test cases similarity of valid pairs. Detailed explanation it is provided in Section 7.3.2

[4]We have implemented the prototype in MATLAB due to its coupling with Simulink, which was the modeling tool used for the simulation of the employed models.

7.3.1 Test Prioritization Criterion

In the second step of the approach (i.e., *Step 2: Prioritize Statically*) we rely on similarity-based criterion to statically prioritize pairs, as it has been found to be effective in previous studies [AHTM+14, FPCY16]. However, any state-of-the-art prioritization criterion [DPC+15, EBA+11, WBA+14] can be employed.

We propose a weight-based criterion to determine the importance of product and test cases similarity when prioritizing pairs. At the Domain level, we prioritize products based on a similarity criterion that adapts the Hamming distance to the context of product line [AHTL+16]. At the Application level, we employ a criterion based on the Euclidean distance.

Similarity at the Domain level

Different works have been published to measure product configurations similarity effectively at the Domain level [AHTM+14, SSRC14a]. We have employed the similarity between product configurations proposed by Al-Hajjaji et al. as it has proven to be effective in previous studies [AHTL+16]. This similarity metric is based on the measurement of selected and deselected features shown in Equation 7.1.

$$PSim(P_u, P_v, F) = 1 - \frac{\mid P_u \cap P_v \mid + \mid (F/P_u) + (F/P_v) \mid}{\mid nf \mid} \tag{7.1}$$

The complete product line set of features is represented with F. When comparing the similarity of two product configurations P_u and P_v, the number of selected $\mid P_u \cap P_v \mid$ and deselected features $\mid (F/P_u) + (F/P_v) \mid$ are taken into account. $PSim$ ranges from 0 to 1, where the lower the $PSim$ value, the higher the similarity of products.

Similarity at the Application level

Previous studies have shown [FPCY16, HAB13, LPBM12] that diverse test cases are more likely to detect faults. We have defined the similarity between test cases with Equation 7.2. The equation relies on the Euclidean distance, one of the most common similarity measure available in the literature. However, any other distance measure such as Hamming, Jaccard, Dice, etc. could have been used [SJH17].

$$TCSim(TC_a, TC_b) = \frac{\sum_{j=1}^{n} D(TC_{a_j} - TC_{b_j})}{n} \tag{7.2}$$

Consider two test cases TC_a and TC_b of the TS test suite. Let us assume that we can abstract a test case as an n-dimensional array. Thus, each test case will consist of an array of n values (e.g., $TC_a = \{a_1, a_2, \ldots, a_n\}$ and $TC_b = \{b_1, b_2, \ldots, b_n\}$). The similarity between the given test cases $TCSim(TC_a, TC_b)$ is measured in Equation 7.2 as the normalized addition of the distances between the array values of each test case. Where $D(TC_{a_j} - TC_{b_j})$ is the Euclidean distance between the j-th value of each test case. Resulting $TCSim$ ranges from 0 to 1, being the lower $TCSim$ values the higher similarity of test cases.

Similarity-based prioritization criterion for pairs

The similarity-based prioritization criterion measures the similarity of the pairs taking into account both the similarity of the products and the similarity of the test cases for the involved pairs. Given two pairs of products and test cases, $PR_1(TC_a, P_u)$ and $PR_2(TC_b, P_v)$, where P_u product is tested with test case TC_a and P_v with TC_b, the Similarity-based Prioritization Criterion ($SbPC$) between the given pairs is measured following Equation 7.3.

$$SbPC(PR_1, PR_2) = PSim(P_u, P_v) \cdot W_p + TCSim(TC_a, TC_b) \cdot W_{tc} \quad (7.3)$$

The product similarity $PSim$ for the given product configurations is measured following Equation 7.1, while similarity between the given test cases is calculated by employing Equation 7.2. Note that W_p is the weight given to the product similarity whereas W_{tc} is the weight given to the similarity between test cases (refer to Section 7.5.2 for details of the values assigned to the weights within the evaluation). W_p and W_{tc} should have values between 0 and 1, moreover, the sum of them should be 1 (i.e., W_p + W_{tc} =1). The resulting measure is the weighted sum of both similarities and ranges between 0 and 1; the lower the weighted similarity, the higher the similarity between pairs.

7.3.2 Similarity-based Prioritization algorithms

The presented approach is supported by two prioritization algorithms. Firstly, the static pair prioritization algorithm, which is employed in the second step of the approach (i.e., *Step 2: Prioritize Statically* in Figure 7.2). Secondly, the dynamic pair prioritization algorithm, which is employed in the third step

multiple times (i.e., *Step 3: Select next Pair (Dynamically Prioritized)* in Figure 7.2).

Static Pair Prioritization algorithm

The *Static Pair Prioritization* algorithm proposes a prioritization of pairs based on similarity. To this end, the algorithm takes all possible combinations of pairs and defines a fixed ordering of pairs according to the test prioritization criterion.

The algorithm employs a pre-processed similarity data set, obtained employing the $SbPC$ test prioritization criterion, that we refer to as Similarity Matrix ($SimMat$). Let $PS = \{P_1, P_2, \cdots, P_{np}\}$ be a product suite with np number of products and $TS = \{TC_1, TC_2, \cdots, TC_{nt}\}$ be a test suite with nt number of test cases. Let us consider that all possible pairs (denoted as $AllPairs$) between products and test cases are *valid pairs*. The number of pairs contained in $AllPairs$ is nPR, where $nPR = np \times nt$. Thus, the similarity matrix $SimMat$ is conformed by calculating the similarity between the nPR pairs of $AllPairs$, producing a $nPR \times nPR$ dimensions matrix.

The algorithm designed for the calculation of static prioritization of pairs is described in Algorithm 5. The *Main* function of the algorithm gets as inputs: (i) the set of valid pairs ($AllPairs$), and (ii) the similarity matrix ($SimMat$). At the beginning, the algorithm selects the first provided pair as prioritized. Next, it iterates with the rest of the pairs to be prioritized, selecting in each case the most dissimilar pair employing the $takeFarthest$ function.

The $takeFarthest$ function receives three input parameters: (i) the current statically prioritized pairs ($StaticPairs$), (ii) the remaining pairs to be prioritized ($AllPairs$) and (iii) the similarity matrix ($SimMat$). The $takeFarthest$ function calculates the similarity between each pair waiting for prioritization and each already prioritized pair. Among all the possible combinations, the pair which is most different from all the statically prioritized pairs is returned. To this end, for each pair to be prioritized, the distance to all previously prioritized pairs is calculated ($distances(k)$). The pair that as a whole is the most different from all the previous ones is reserved ($dist(j)$). Among all reserved pairs, the most different one is selected ($nextStaticPair$). As a result, the algorithm provides the statically prioritized set of pairs ($StaticPairs$).

Algorithm 5: Static Pair Prioritization

input : – *AllPairs*, Set of all valid pairs
– *SimMat*, similarity matrix
output : – *StaticPairs*, Set of all pairs statically prioritized

```
1  Function Main
2      StaticPairs(1) ← AllPairs(1) ;                    // place 1st pair
3      for i=2 to nPR do
4          nextStaticPair ← takeFarthest(StaticPairs,AllPairs,SimMat);
5          StaticPairs(i) ← nextStaticPair;
6          AllPairs(i) ← null ;        // Remove nextStaticPair from AllPairs
7      end
8  End Function
9  Function takeFarthest (StaticPairs,AllPairs,SimMat)
10     nAllPairs ← size(AllPairs) ;                    // # remaining Pairs
11     nStaticPairs ← size(StaticPairs) ;              // # prioritzed Pairs
12     for j=1 to nAllPairs do
13         for k=1 to nStaticPairs do
               // get distances between AllPairs and StaticPairs items
14             distances(k)=SimMat(AllPairs(j),StaticPairs(k));
15         end
16         dist(j) ← min(distances);
17     end
18     nextStaticPair ← max(dist);
19 return nextStaticPair
```

Dynamic Pair Prioritization algorithm

To efficiently conduct steps of the third phase, the Dynamic Pair Prioritization algorithm was designed. The underlying idea of the algorithm consists of the reallocation of pairs according to the product fault detection capability. We focus on product fault detection capability because we hypothesize that a product where faults are being detected is more likely to have other faults. The algorithm is configured with two parameters *nStartUp* and *nRealloc*: the *nStartUp* parameter is used to configure the number of pairs that will initially be executed to feed the Test History. The *nRealloc* parameter is used to configure the number of pairs that will be taken to be dynamically reordered. Hereafter an example is employed to describe the detailed functioning of the algorithm. Additionally, Algorithm 6 provides the pseudo-code of the Dynamic Pair Prioritization algorithm.

Algorithm 6: Dynamic Pair Prioritization

input : – *nStartUp*, number of pairs initially executed to feed Test History
– *nRealloc*, number of pairs to be reallocated in each iteration
– *StaticPairs*, set of all pairs statically prioritized

output : – *DynamicPairs*, set of all pairs dynamically prioritized

```
   // Stage 1 - Execute and allocate initial nStartUp number of pairs
 1 for i=1 to nStartUp do
 2 |   verdict = executePairs(StaticPairs(i));
 3 |   updateTestHistory(verdict);
 4 |   DynamicPairs(i) ← StaticPairs(i);
 5 end
   // Stage 2 - Reallocate remaining pairs dynamically
 6 for i=nStartUp+1 to length(StaticPairs) do
 7 |   RemainingPairs(i) = StaticPairs(i);
 8 end
 9 for i=1 to length(RemainingPairs) do
   |   // select next nRealloc number of pairs
10 |   for j=i to i+nRealloc-1 do
11 |   |   pairs2Realloc(j) ← RemainingPairs(j);
12 |   end
13 |   if product of pairs2Realloc(1) never executed then
14 |   |   verdict = executePair(pairs2Realloc(1));
15 |   |   updateTestHistory(verdict);
16 |   |   DynamicPairs(i) ← pairs2Realloc(1);
17 |   else
18 |   |   highestFDCIndex ← getHighestFDCIndex(pairs2Realloc);
   |   |   // Place highest productsFDC pair in first position
19 |   |   pairs2ReallocSorted(1) ← pairs2Realloc(highestFDCIndex);
20 |   |   pairs2Realloc(highestFDCIndex) ← null ; // remove
   |   |   // Place remaining pairs2Realloc pairs
21 |   |   for j=2 to nRealloc do
22 |   |   |   pairs2ReallocSorted(j) ← pairs2Realloc(j-1) ;
23 |   |   end
24 |   |   verdict = executePair(pairs2ReallocSorted(1));
25 |   |   updateTestHistory(verdict);
26 |   |   DynamicPairs(i) ← pairs2ReallocSorted(1);
   |   |   // Update RemainingPairs with reallocated pairs
27 |   |   for j=i to i+nPairs2Realloc-1 do
28 |   |   |   RemainingPairs(j) ← pairs2ReallocSorted(j);
29 |   |   end
30 |   end
31 end
```

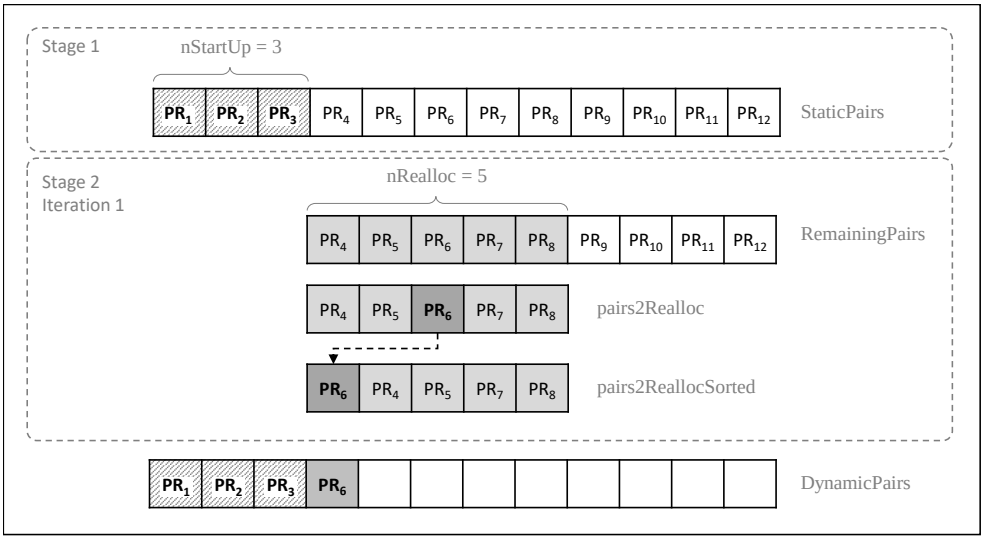


Figure 7.3: Graphical representation of the Dynamic Pairs Prioritization Approach example, where the $nStartUp$ pairs (PR_1, PR_2 and PR_3 shadowed with diagonal lines) are placed into the $DynamicPairs$ output during the first stage. The first iteration of the second stage is also represented, where the pair with highest priority (PR_6 emphasized with dark gray shadow) is selected to be placed into the $DynamicPairs$ output.

Let us consider the motivating example of Figure 7.3 where a set of twelve statically prioritized pairs is provided $StaticPairs = \{PR_1, PR_2, \cdots, PR_{12}\}$. In the first stage of the dynamic pair prioritization algorithm (detailed between lines 1 to 5 of Algorithm 6) the first $nStartUp$ number of pairs of the $StaticPairs$ are selected (i.e., pairs PR_1, PR_2 and PR_3 shadowed with diagonal lines in Figure 7.3). The selected pairs are placed at the very first positions of the dynamically prioritized pairs algorithm output $DynamicPairs$ $= \{PR_1, PR_2, PR_3\}$. The selected pairs are next executed and Test History is accordingly updated. Considering that the dynamic pair prioritization algorithm is based on historical information (i.e., test executions verdicts are considered to calculate the fault detection capability of products), this first stage consists of the selection and execution of a number of pairs, guaranteeing a minimum of data in the Test History. This is because it has been found in previous studies, that history-based techniques require a start-up phase in order to be effective [KIJT18].

In the second stage, the remaining pairs are iteratively processed and dynamically reallocated. To this end, at the beginning of the first iteration, remaining pairs are taken, i.e., pairs not selected in the first stage are taken and placed into $RemainigPairs$ (detailed between lines 6 to 8 of Algorithm

6). Next, the first $nRealloc$ number of pairs are taken from $RemainingPairs$ to be reallocated and placed into $pairs2Realloc$ (lines 10 to 12). Following the example, next five pairs are taken (i.e., $pairs2Realloc = \{PR_4, PR_5, PR_6, PR_7, PR_8\}$). Fault detection capability of products related to $pairs2Realloc$ pairs are taken into account to reallocate pairs (line 18 details how the index of pair with highest FDC is obtained, i.e., $highestFDCIndex$). As a result, the pair with the highest capability to detect faults is placed at first position of $pairs2ReallocSorted$.

Remaining pairs are placed at the $pairs2ReallocSorted$ following the $pairs2Realloc$ ordering (lines 21 to 23). To continue with the example, let us suppose that PR_6, emphasized with dark gray shadow, obtains highest product fault detection capability (i.e., $productFDC$) among pairs contained in $pairs2Realloc$, therefore it is placed in the first position of $pairs2ReallocSorted$. Next, PR_4, PR_5, PR_7 and PR_8 are placed just after PR_6, resulting $pairs2ReallocSorted = \{PR_6, PR_4, PR_5, PR_7, PR_8\}$. The first pair of $pairs2ReallocSorted$ (i.e., PR_6) is placed at $DynamicPairs$, it is executed and Test History is updated (lines 24 to 26). At the end of the first iteration, $DynamicPairs$ is composed of four pairs, $DynamicPairs = \{PR_1, PR_2, PR_3, PR_6\}$. After each iteration $RemainingPairs$ is updated removing the last prioritized pair (lines 27 to 29). In our example PR_6 is removed from $RemainingPairs$. In subsequent iterations, $nRealloc$ number of pairs are taken and sorted, to select the pair with the highest fault detection capability. Once all iterations are completed, $DynamicPairs$ is provided with all prioritized pairs as algorithm output.

7.4 Application of the Approach on Configurable Simulation Models

This section describes how the Dynamic Prioritization approach proposed in Section 7.3 is adapted to the domain of configurable simulation models.

7.4.1 Test Prioritization Criteria for Configurable Simulation Models

Several metrics exist to measure the quality of test cases in the context of simulation models, which can be used to prioritize tests. These metrics can be classified either as black-box (i.e., those focusing on the inputs and outputs of the simulation models) or white-box (i.e., those measuring internal white-box

coverage). In our previous work [AWA+18, AWM+19] we proposed a set of black-box metrics and demonstrated that their performance was better than traditionally employed white-box metrics. In a later study, we adapted both black-box and white-box metrics to measure the quality of tests in the context of configurable simulation models [MAES19b]. For this study, among the metrics proposed in [MAES19b], we opted to use *input-based test similarity*.

We opted to use this as adequacy criterion for prioritizing tests in the context of configurable simulation models due to two main reasons. Firstly, is that previous studies have shown that diverse test cases are more likely to detect faults [FPCY16, HAB13], and these have been successfully used as criteria to prioritize test cases both in general purpose software systems [FCWZ14, MCVB18, NH15, THHB14], as well as in the context of product line engineering at the Domain engineering level [AHTM+14]. The second reason is that, unlike output-based test similarity, or other quality metrics proposed in [AWA+18, AWM+19, MAES19a], input-based test similarity does not require tests to be executed beforehand. Subsequently, our approach can be used at early validation stages right after the tests have been generated.

Similarity at the Domain level for Configurable Simulation Models

Similarity at Domain level does not require specific adaptation for configurable simulation models context. Therefore, the proposal made in Section 7.3.1 to measure similarity between products can be directly employed.

Similarity at the Application level for Configurable Simulation Models

In order to adapt the similarity at Application level to configurable simulations models we relied on the normalized Euclidean distance of signals which has already been used in previous works within the model-based testing context [MNBB15, MNBB16]. Specifically, We employed the normalized Euclidean distance proposed in [AWA+18] to tackle the diversity of simulation-time lengths between test cases.

$$D(sig_a, sig_b) = \frac{\sqrt{\sum_{i=0}^{min(k_{sig_a}, k_{sig_b})} (sig_a(i \cdot \Delta t) - sig_b(i \cdot \Delta t))^2}}{\sqrt{K+1} \cdot (maxIsig_a - minIsig_a)} \tag{7.4}$$

Given two signals sig_a and sig_b related to inputs from two different TC_a and TC_b test cases, Equation 7.4 shows how the test-length adapted Euclidean distance between aforementioned signals $D(sig_a, sig_b)$ is calculated. Resulting

distance values range between 0 and 1, where the higher the distance value is, the more dissimilar the two signals are.

Being k_{sig_a} and k_{sig_b} the number of observed simulation steps of the signals sig_a and sig_b respectively, the expression $min(k_{sig_a}, k_{sig_b})$ is the number of steps of the signal whose test case has a lower Test Execution Time (TET). Among all vector values that the input signal $Isig_a$ can acquire during the simulation, $maxIsig_a$ and $minIsig_a$ are the maximum and minimum values respectively. K is the number of observed simulation steps that the test case with the highest TET in the whole test suite has. Note that the distance between the two signals will be the same regardless of the order in which it is calculated (i.e., $D(sig_a, sig_b) = D(sig_b, sig_a)$).

Three groups of signals must be considered to compare two test cases executed in two product configurations: (i) the signals that both configurations share, (ii) the signals that are present only in one configuration but not in the other and (iii) the signals that both configurations do not have.

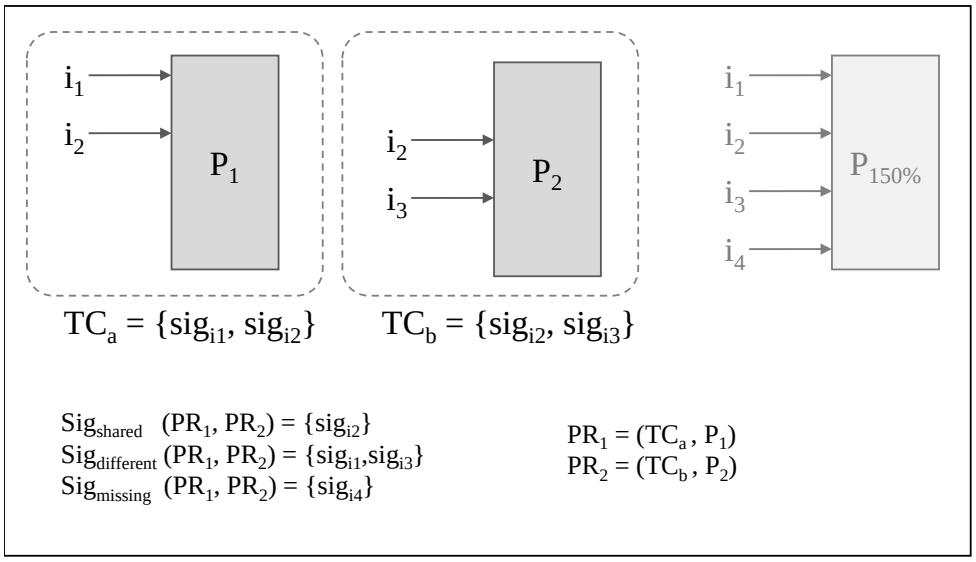


Figure 7.4: Illustrative example of the three groups of signals on which the Signal-based Similarity measure is founded. Test cases TC_a and TC_b are executed respectively with products P_1 and P_2. When both executions of pairs are compared, sig_{i2} is the only shared signal (Sig_{shared}), while sig_{i1} and sig_{i3} are both present in only one of the executions but not in the other ($Sig_{different}$). Finally, sig_{i4} is missing from both executions ($Sig_{missing}$).

Let us consider the example depicted in Figure 7.4 to describe the three different groups of signals. Suppose two product configurations P_1 and P_2 of the product suite $SMPS$. Product P_1 contains inputs $i1$ and $i2$, while product

P_2 contains inputs $i2$ and $i3$. In addition, neither of the two products contains the $i4$ input available in other $SMPS$ products. We also have two test cases TC_a and TC_b executed respectively in P_1 and P_2 (i.e., pairs $PR_1 = (TC_a, P_1)$ and $PR_2 = (TC_b, P2)$). The TC_a will be adapted to product P_1 to contain two signals for inputs $i1$ and $i2$, $TC_a = \{sig_{i1}, sig_{i2}\}$. TC_b will be adapted to P_2, $TC_b = \{sig_{i2}, sig_{i3}\}$.

In this example, sig_{i2} is the only *shared* signal between TC_a and TC_b test cases, while sig_{i1} and sig_{i3} are signals that are present in only one of the products (*different*). Finally, sig_{i4} is the signal *missing* in both products.

Taking into account the different sets of signals that test cases may have, when adapting the application similarity measure to the context of configurable simulation models, we propose two measures to compute the similarity between test cases.

Grouped-Signals based Test Case Similarity measure (GSbSim) This proposed measure denoted as $GSbSim$ provides similarity between test cases taking into account the three different groups of signals described in Section 7.4.1. Note that the test cases employed in this measure are already adapted to specific products. Equation 7.5 measures the $GSbSim$ similarity.

$$GSbSim(TC_a, TC_b) = \frac{\sum_{j=1}^{s} D(sig_{aj}, sig_{bj}) + \sum_{j=1}^{d} maxD}{nt} \tag{7.5}$$

The number of shared signals is represented by s, while $D(sig_{aj}, sig_{bj})$ refers to the test-length adapted Euclidean distance between the j signals of test cases a and b. The number of different signals is represented by d. These different signals are present only in one of the configurations but not in the other and therefore, we apply the maximum distance value $maxD = 1$. For missing signals, the minimum distance is applied $minD = 0$, which makes incorporating it into the equation unnecessary. The similarity between two test cases can obtain values between 0 and 1, so that $GSbSim(TC_a, TC_b) = 0$ means that the test cases are identical, and completely different if the obtained value is 1.

All-Signals based Test Case Similarity measure (ASbSim) This measure, unlike the previous one, does not adapt test case input signals to product inputs. Let TS_{150} be the suite containing $NIsig$ number of input signals. To obtain the similarity between two test cases TC_a and TC_b of TS_{150} test suite without taking into account product-specific input signal adaptations,

the $NIsig$ signal distances are averaged as indicated in Equation 7.6 (i.e., all signal distances are employed to calculate the similarity between test cases, regardless of the number of signals of the product in which they are executed).

$$ASbSim(TC_a, TC_b) = \frac{\sum_{j=1}^{NIsig} D(sig_{aj}, sig_{bj})}{NIsig} \tag{7.6}$$

The similarity is obtained taking into account all signals (i.e., $NIsig$ number of signals) and distances are measured according to the test-length adapted Euclidean distance equation. The similarity between two test cases can obtain values between 0 and 1, meaning $ASbSim(TC_a, TC_b) = 0$ that the test cases are identical, and completely different if the obtained value is 1.

Similarity-based prioritization criteria for Configurable Simulation Models

The similarity-based prioritization criterion described in Section 7.3.1 is split into two criteria when adapting it to the context of configurable simulation models. Thus, for each test case similarity measure (i.e., $GSbSim$ and $ASbSim$) the criterion is adapted.

Grouped-Signals Similarity of pairs (GS) The *Grouped-Signals Similarity of pairs* criterion measures the similarity of pairs of products and test cases taking into account the Grouped-Signals based Similarity measure ($GSbSim$). Given two pairs of products and test cases, $PR_1(TC_a, P_u)$ and $PR_2(TC_b, P_v)$, where P_u product is tested with test case TC_a and P_v with TC_b, the Grouped-Signals Similarity (GS) between given pairs is measured following Equation 7.7.

$$GS(PR_1, PR_2) = PSim(P_u, P_v) \cdot W_p + GSbSim(TC_a, TC_b) \cdot W_{tc} \tag{7.7}$$

Resulting measure is the weighted sum of both product ($PSim$) and test cases ($ASbSim$) similarities and ranges between 0 and 1; the lower the weighted similarity, the more identical the pairs.

All-Signals Similarity of pairs (AS) The *All-Signals Similarity of pairs* criterion measures the similarity of pairs of products and test cases taking into account the All-Signals based Similarity measure ($ASbSim$). As in the case of GS, the All-Signals Similarity (AS) is measured following Equation 7.8, however, $ASbSim$ measure is employed .

$$AS(PR_1, PR_2) = PSim(P_u, P_v) \cdot W_p + ASbSim(TC_a, TC_b) \cdot W_{tc} \quad (7.8)$$

Resulting measure is the weighted sum of both product ($PSim$) and test cases ($ASbSim$) similarities and ranges between 0 and 1; the lower the weighted similarity, the more identical the pairs.

7.4.2 Similarity-based prioritization algorithms for Configurable Simulation Models

This section presents how the two algorithms (i.e., static and dynamic) for test prioritization of pairs are adapted to the configurable simulation models context. Considering that the two test prioritization criteria defined in Section 7.4.1 can be applied to both algorithms, by the end of this section a total of four variants of the algorithms are presented (i.e., two variants per algorithm depending on the employed test prioritization criterion).

Static Pair Prioritization algorithm for Configurable Simulation Models

When adapting the static pair prioritization algorithm, it must again be taken into account that in the context of configurable simulation models, two prioritization criteria are available to calculate the weighted similarity of pairs (i.e., GS and AS). Thus, two variants of the similarity matrix ($SimMat$) are calculated: the $SimMat_{GS}$ similarity matrix calculated employing the GS criterion for the SGS algorithm variant; the $SimMat_{AS}$ similarity matrix that uses the AS criterion for the SAS algorithm variant.[5] Note that only one of the two different similarity matrices is provided as input to the algorithm, depending on the intended algorithm variant (i.e., $SimMat_{GS}$ or $SimMat_{AS}$).

Dynamic Pair Prioritization algorithm for Configurable Simulation Models

The Dynamic Pair Prioritization algorithm described in Section 7.3 can be applied to any product line regardless of how it is implemented (software-based, model-based, etc.). Therefore, when it is applied to the configurable simulation models, it only requires considering the measure that has been employed to statically prioritize the pairs. That is, the Dynamic Prioritization

[5]DISAMBIGUATION: We would like to differentiate the SAS SPL testing strategy described in Section 3.1 from the variant of the static prioritization algorithm using the AS metric presented in this chapter, which also has the SAS acronym.

Algorithm receives as input the set of statically prioritized pairs (i.e., step 2 of the approach described in Figure 7.2), which can be calculated using any of the GA or AS test prioritization criteria. Therefore, the variant of the algorithm Dynamic Grouped-Signals Similarity (DGS) is defined, if the GS criterion is used to obtain the statically prioritized input pairs. On the other hand, if AS has been used to statically prioritize the input pairs, the variant of the algorithm Dynamic All-Signals Similarity (DAS) is defined.

7.5 Evaluation

7.5.1 Research Questions

We defined the following experiment to assess the behavior of the proposed test prioritization algorithms. With the first Research Question (RQ1) we intended to perform a sanity-check to assess whether the proposed algorithms performed better than the baseline technique. A test prioritization technique that minimizes input signals diversity was employed as a baseline, as it was proposed in [HPH+16]. The second research question (RQ2) aimed at comparing the dynamic test prioritization algorithms with respect to the static ones. Thirdly, we wanted to assess which of the proposed test prioritization criteria for prioritizing test cases in the context of configurable systems performed better (RQ3). The dynamic test prioritization algorithm has a set of user-predefined configuration options. Our objective for RQ4 was to study how these parameters (i.e., the size of the test cases executed for the start-up and the size of number of test cases to be reallocated) affected the performance of the algorithm. Lastly, by using information of the mutants and their relation of detection by each test case, we were able to obtain the optimal and worst-case test prioritization cases. RQ5 aimed at comparing an upper and lower bound comparison with respect to our algorithms. We raise, thus, the following five RQs:

- **RQ 1 – Sanity check:** Is the performance of the proposed algorithms better than that of the baseline technique?
- **RQ 2 – Dynamic vs. static test prioritization:** Which of the proposed algorithms best addresses the test prioritization problem?
- **RQ 3 – Test prioritization criteria:** Which of the proposed measures best contributes to test case prioritization in the context of configurable simulation models?
- **RQ 4 – Algorithms configurations:** How does the configuration of the dynamic algorithm affect the performance of the proposed algorithms?
- **RQ 5 – Upper and Lower bound comparison:** How do the best test prioritization algorithms compare with the optimal and worst-case test prioritizations?

7.5.2 Experimental Setup

This section explains how we designed the experiments to answer the five RQs.

Case Studies

The proposed test prioritization approach was evaluated using two case studies: (1) the *Industrial Tanks* (IT) case study and (2) the *Car Windows* (CW) case study, further detailed in Sections 4.4.2 and 4.4.2 respectively. Each consists of a Highly-Configurable Cyber-Physical System (HCCPS) from two different domains (i.e., industry automation domain and automotive domain). The *Industrial Tanks* case study provides a control system for a tank product family, where it aims at controlling the liquid level of each tank according to certain constraints. As for the *Car Windows* case study, the system controls different windows of a car. Not having any information about real faults, mutation testing was employed, which has been found to be a good substitute of real faults [JJI+14] (Refer to Section 7.5.2 for more information). The characteristics of the case studies are shown in Table 7.1, while the characteristics of the generated products, test suite and mutants for the evaluation are presented in Table 7.2.

Table 7.1 summarizes the key characteristics of each case study used in our experiment. A **feature model** was designed to describe the variability of each product family. In the *Industrial Tanks* case the feature model was composed of 36 features and 7 constraints, while 30 features and 13 constraints

Table 7.1: Key characteristics of the selected case studies. The *Blocks* column provides the number of Simulink blocks for each of the case studies, while *Inputs* and *Outputs* columns provide the number of input and output ports per case study. The *Features* and *Constraints* columns specify for each case study the number of features and constraints for the corresponding feature model.

Case Study	Blocks	Inputs	Outputs	Features	Constraints
Industrial Tanks (IT)	306	18	3	36	7
Car Windows (CW)	227	18	4	30	13

were employed for the Car Windows case. Taking into account these feature models, thousands of products could be configured, specifically, 39,582 and 11,824 product configurations for IT and CW cases respectively. For the experiment, we derived a total of 17 (IT) and 28 (CW) products by using the ICPL algorithm [JHF12] and the pairwise criterion. Furthermore, according to Kuhn et al., empirical investigations have concluded that from 50 to 97% of software faults could be identified by pairwise combinatorial testing [KKLH09]. In addition, employing a higher-strength t-wise approach (e.g., 3-wise) would considerably increase the test execution time, and the pay-offs are not that high. The ICPL algorithm was employed because it is an algorithm that is integrated in the *FeatureIDE* tool and is fast enough to use it in practice. However, any product sampling strategy can be used.

MATLAB/Simulink was used to implement the **150% model** in order to represent the variability of each product line, as it is the de facto tool for modeling and simulating complex dynamic models (e.g., CPSs) [BNSB16]. Refer to Section 4.4.1 for further details related to 150% simulation models. The number of blocks for each of the models was of 306 and 227. Both models were designed with 18 input ports for system stimulation. In addition, the *Industrial tanks* case generated 3 output port signals while the *Car Windows* case generated 4 output port signals to report system status.

In our context, a **test case** consists of a number of signals that stimulates the System Under Test (SUT). This type of test cases have been previously used in similar works [AWA+18, MNBB16, MNBB19]. We adapted a Simulink-based tool for test generation [MNBB19] in order to randomly generate test cases for the 150% Simulink models. These test cases later needed to be instantiated for each of the derived products (i.e., for each product, we needed to remove some of the signals from the original 150% test cases). Note that in our case studies, each generated test case was valid for all derived configurations. In

Table 7.2: Derived products, generated test cases and mutants of selected case studies. The *Derived Products* column provides the number of product generated with pair-wise technique per case study. The following three columns describe the criterion employed to generate test cases, the number of test cases generated for the 150% model and the total number of test cases after adapting 150% test cases to each derived product. The last two columns show the number of mutants per case study; the first column shows the total number of initially generated mutants, while the second column refers to the final number of mutants employed in the evaluation after applying mutant selection.

Case Study	Derived	Test Cases			Number of Mutants	
	Products	Gen.Criterion	150%	Total	Initial	Final
Industrial Tanks	17	Random	150	2550	246	30
Car Windows	28	Random	150	4200	209	36

total we generated 150 test cases per case study, and by combining them with each of the derived products, we had in total 2,500 and 4,200 test cases for each of the case studies (i.e., 150 test cases $\times$ 17 product = 2,550 test cases for the Industrial Tanks case study and 150 test cases $\times$ 28 products = 4,200 test cases for the Car Windows case study). The selected number of test cases are in-line with other approaches, which target industrial case studies. Within an industrial case study, Wang et al. used 138 to 239 test cases [WAG15]. In another industrial setting, Hajri et al., had test suites of 83 to 113 test cases [HGPB20]. In our previous study we used up to 120 test cases, also including an industrial case study [AWSE19].

Evaluation metric

The Average Percentage of Faults Detected (APFD) was employed to measure the fault detection rate of the selected test prioritization approach, as it is the most widely used evaluation metric in the context of test case prioritization [CM13]. This metric measures the weighted average of the percentage of faults detected over the life of a test suite [CM13]. Let T be a test suite containing n test cases, and F be a set of m faults revealed by T. Let T' be a specific ordering of test cases from T, where TF_i takes the first position revealing i fault. Equation 7.9 provides the APFD metric for the T' test suite, where APFD values ranges from 0 to 1. The closer the APFD value of a test suite is to 1, the faster it will detect faults.

$$APFD(T') = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \cdot m} + \frac{1}{2n} \tag{7.9}$$

Not having any information about real faults, mutation testing was employed, which has been found to be a good substitute of real faults [JJI+14]. Within this technique, a version of the original system (mutant) is created and a fault (mutation) injected. A mutant is considered killed by a test case if its outputs differ with respect to the original model. To design mutants, we employed mutation operators proposed by Hanh et al. for MATLAB/Simulink models [HBT16]. Therefore, a certain number of mutants were initially generated per case study, as reported in table 7.2. In addition, each fault was related to a specific feature associated to the 150% model. We injected 246 and 209 mutants into the 150% model of the Industrial Tanks and Car Windows cases respectively.

When a specific product was derived, a 100% product model was created including one copy of the original system and a number of mutants. The number of mutants included in a specific product was defined by the selected product configuration, where those mutants related to features (or feature interactions) not included in the product configuration were removed. Generated test cases (i.e., 2,550 and 4,200 test cases of IT and CW case studies) were next executed with respective product models (which include mutants) and results analyzed to gather mutant detection capability of each test case.

To obtain the final set of mutants for the evaluation, three mutant selection criteria were applied: firstly, undetectable mutants were removed (i.e., mutants that were not detected by any test case), in order to avoid the inclusion of equivalent mutants. Secondly, duplicated mutants were removed (i.e., mutants equivalent to one another but not to the original model), as recommended by Papadakis et al [PJHLT15]. Lastly, we removed mutants that were easily detected by test cases. Subsequently, the remaining number of mutants for the evaluations was reduced from 246 to 30 for the *Industrial Tanks* case study and from 209 to 36 for *Car Windows*. Although the number of mutants employed in the evaluation is not large compared to other approaches, it should be noted that MATLAB/Simulink models are expensive to run. In this context, the number of mutants proposed in this approach is similar to or higher than other studies with MATLAB/Simulink [AWSE16a, NOM06, MNBB15, MNBB19, MASE17].

Experiment Runs

To evaluate the proposed approach, we generated different experimental scenarios. To this end, for each case study we first defined different test suite sizes. Specifically, we created 10 test suite sizes from 50 to 140 test cases,

adding 10 test cases at a time (i.e., test suite sizes of 50, 60, 70,...,140 test cases). Subsequently, we chose 50 different test suites (i.e., a collection of test cases) per test suite size by randomly selecting test cases from the initially generated 150 test cases. Different number of test cases were selected because each test case is different, and therefore, we wanted to evaluate our approach under different conditions (e.g., test cases that are good at detecting faults, test cases that are not that good, etc.). By combining the 10 test suite sizes generated with the 50 test case selections for each test suite size, we obtained a total of 500 experimental scenarios for each case study.

For each experimental scenario, the selected algorithms were executed and the APFD obtained. The static prioritization algorithm variants as well as the baseline technique were executed once per scenario. Nevertheless, to find the answer to RQ4, we ran one execution per algorithm parameters configurations for dynamic prioritization algorithm variants. As described in Section 7.3, the dynamic prioritization algorithm consists of two configuration parameters. The $nStartUp$ parameter determines the number of pairs (of products and test cases) that are executed to start-up, while the $nRealloc$ parameter defines the number of pairs the algorithm takes to reallocate.

We set up five values for $nStartUp$ parameter and six values for $nRealloc$ parameter, after a number of preliminary runs. Specifically, $nStartUp = \{1, 5, 10, 20, 50\}$ and $nRealloc = \{5, 10, 20, 50, 100, 200\}$ were selected. Thus, the dynamic algorithm was executed 30 times per experimental scenario, which yielded a total number of 15,000 executions per case study. As for the configurations of the test prioritization criteria, note that we did not vary the weights provided to the similarity of the products (i.e., $W_p = 0.5$) nor the similarity of the test cases (i.e., $W_{tc} = 0.5$). Thus, when calculating the test prioritization criteria, the weight values were the same, giving the same importance to both test prioritization criteria (applied in both static and dynamic prioritization algorithms).

We developed a script to execute the experiment as it is shown in the following pseudo code 7.

Additionally to the loops delimiting the experimental scenarios, and complementary runs for the dynamic algorithm, the algorithm script also calculated the data sets that the algorithms employed in their calculations. Specifically three types of data sets were calculated: (i) the data set of similarity between products per case study, (ii) the data set of similarity between test cases and (iii) the similarity matrices per experimental scenario.

Algorithm 7: Experiment execution pseudo-code

```
1  for each case study do
2      Calculate Product Similarity Dataset;
3      for TestSuiteSize=50 to 140 do
4          for TCSelection=1 to 50 do
5              Calculate Test Case Similarity Dataset;
6              Calculate Similarity Matrix;
7              for each algorithm do
8                  if algorithm is Dynamic then
9                      for each Dynamic Parameters Configuration do
10                         Execute and get metrics;
11                     end
12                 else
13                     Execute and get metrics;
14                 end
15             end
16         end
17     end
18 end
```

7.5.3 Results and Analysis

We now analyze the results obtained for the experiment.

RQ1 – Sanity Check

RQ1 aimed at comparing the proposed test prioritization algorithms with respect to the baseline technique. Similar to Henard et al., [HPH+16], the baseline test prioritization technique prioritized test cases statically by aiming at minimizing the similarity distance among test cases. For the static approaches as well as for the 500 test scenarios, we compared how many of the selected test prioritization algorithms were performing better than the baseline technique, and measured them in percentages. In the case of the dynamic test prioritization approaches, we had 30 different configurations, and thus, in total 15,000 potential comparisons. For these 15,000 comparisons, we also measured the percentage of times a technique performed better than another one.

Table 7.3 summarizes these results; the columns A and B show the selected test prioritization techniques. Later, there are three sub-columns for each of the case studies (i.e., IT and CW). Column A > B shows that the percentage of times the algorithm in column A performed better than the algorithm in column B in terms of APFD. Column A < B on the other hand, shows

the contrary. Column A = B shows the number of times both algorithms showed the same APFD. Clearly, the selected test prioritization algorithms outperformed the baseline technique for all selected scenarios. These results can also be corroborated by means of descriptive statistics shown in Figures 7.5 and 7.6, where the distribution of the 500 test scenarios for each technique is depicted.[6] This means that the selected algorithms as well as the test prioritization criteria are performing correctly, and that the test prioritization problem is not trivial to solve in this context.

When considering the average APFD values, for the IT case study, the SGS technique outperformed the baseline by 28%, the SAS technique by 30%, the DGS by 28% and the DAS by 31%. As for the CW case study, all of them outperformed the baseline algorithm by 11% when considering the average APFD values.

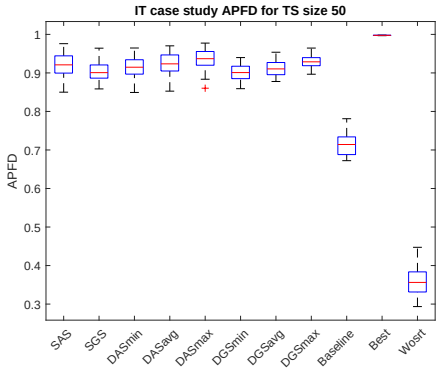
Table 7.3: RQ1: APFD percentage comparison between the proposed test prioritization algorithms (i.e., SGS, SAS, DGS and DAS) with respect to the baseline technique.

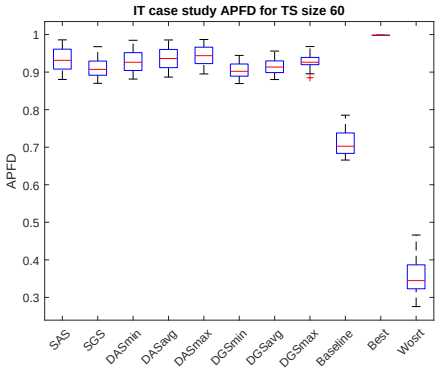
A	B	IT			CW		
		A >B	A = B	A <B	A >B	A = B	A <B
SGS	Baseline	**100%**	0%	0%	**100%**	0%	0%
SAS	Baseline	**100%**	0%	0%	**100%**	0%	0%
DGS	Baseline	**100%**	0%	0%	**100%**	0%	0%
DAS	Baseline	**100%**	0%	0%	**100%**	0%	0%

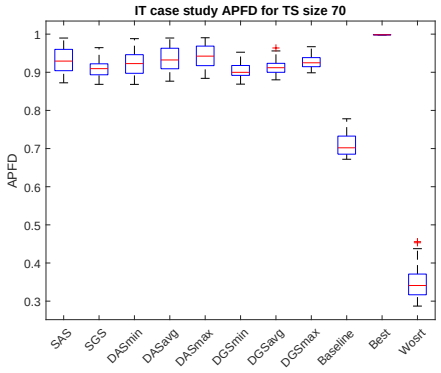
RQ2 – Dynamic vs Static Test Prioritization

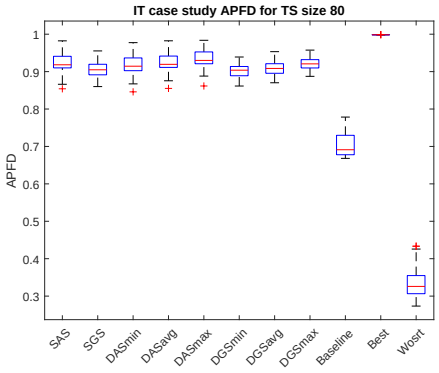
The second research question aimed at comparing the dynamic test prioritization approaches with respect to traditional static prioritization approaches. Results are summarized in Table 7.4. Similar to RQ1, we compare for the 500 test scenarios and 30 configurations of the dynamic test prioritization algorithm, how many times the static algorithm outperformed the dynamic one (column A > B); how many times both performed equally (column A=B); and how many times the dynamic algorithm performed better than the static one (column A<B), all of them in terms of APFD. Note that in this case we are comparing how the approaches work algorithmically, and thus, to make the

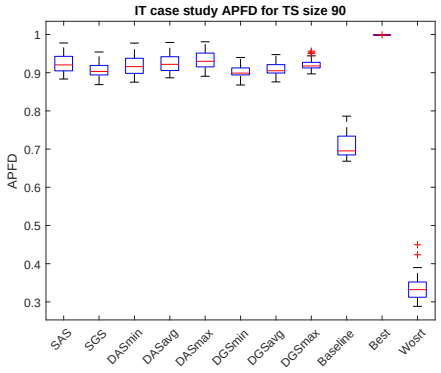
[6] To make the box plots of a reasonable size, the results for the dynamic test prioritization algorithms are shown as minimum (i.e., worst configuration), average, and maximum (i.e., best configuration) values for each test scenario

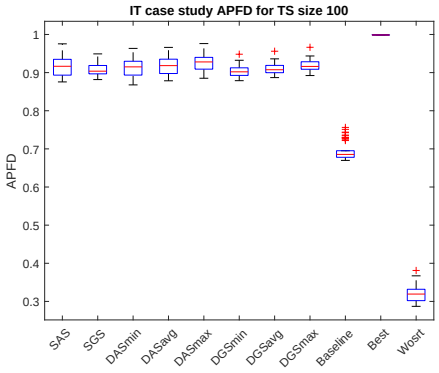

IT case study APFD for TS size 50
APFD
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt

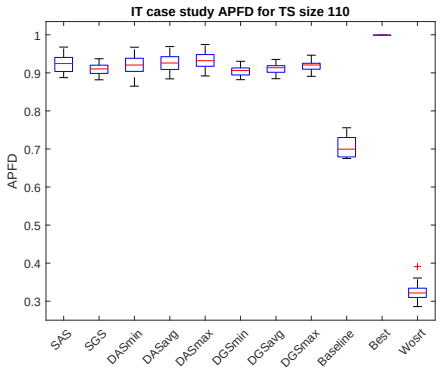

IT case study APFD for TS size 60
APFD
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt

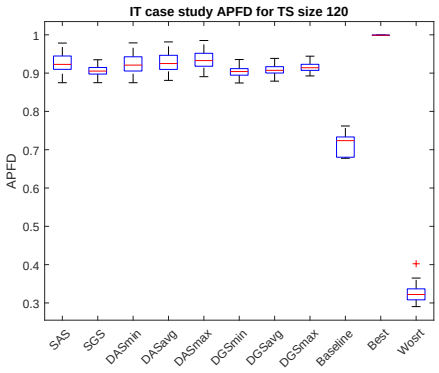

IT case study APFD for TS size 70
APFD
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


IT case study APFD for TS size 80
APFD
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


IT case study APFD for TS size 90
APFD
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


IT case study APFD for TS size 100
APFD
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


IT case study APFD for TS size 110
APFD
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


IT case study APFD for TS size 120
APFD
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt

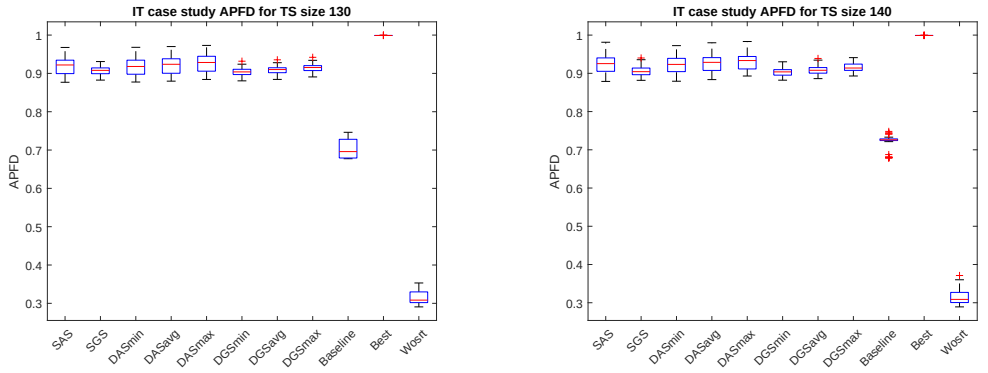


Figure 7.5: Distribution of test scenarios per technique and test suite for IT case study. Note that for the dynamic techniques results are shown as minimum, average and maximum to make the charts of a reasonable size.

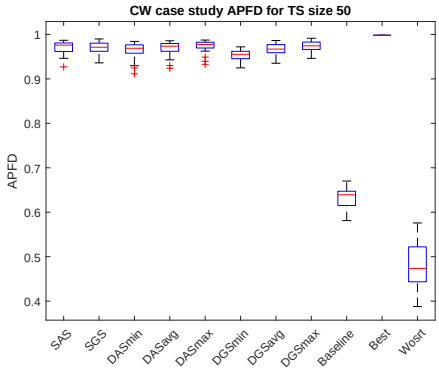
comparison fair, we make the comparison between the algorithms maintaining the same test prioritization criterion.

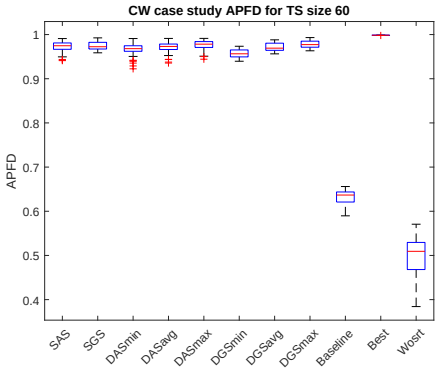
In both case studies, the dynamic test prioritization algorithm outperformed the static one. In the IT case study, the dynamic test prioritization algorithms outperformed the static algorithms for 78.9% and 80.7% of test scenarios. In the case of the CW case study, this percentage dropped a bit, to 53.3% and 59.3% of test scenarios. On the contrary, the static prioritization outperformed the dynamic test prioritization approaches for 19.2% and 20.8% of test scenarios and 46.3% and 40.4% of test scenarios for the CW case study. Even though in the CW case study the number of test scenarios where the static approaches outperformed the dynamic ones in terms of APFD is quite high, the dynamic approaches still showed better performance.

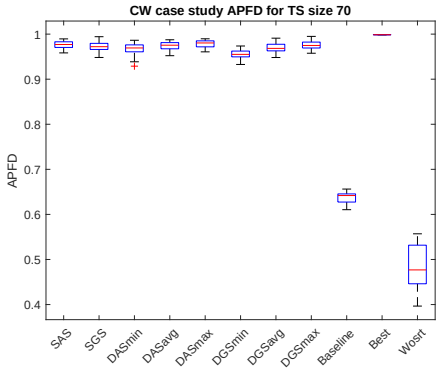
Interestingly, even the dynamic approaches outperformed the static ones in numbers, the improvement over the static one when taking into account the average values of the APFDs were not high. For the IT case study, the DGS outperformed the SGS by 0.37% and the DAS the SAS by 0.21%. A potential reason for this could be that with large test suites the APFD measurement might not be very meaningful [WSKR06]. Conversely, for the CW case study, the average APFD values for the static techniques were slightly higher than the dynamic ones, although the differences were negligible.

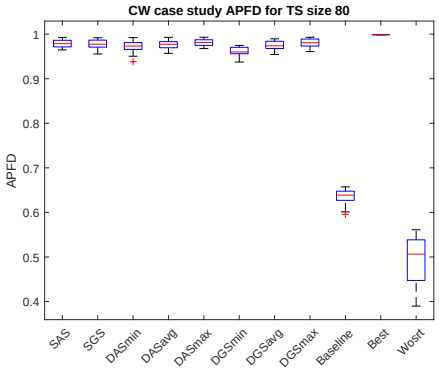
RQ3 – Test Prioritization Criteria

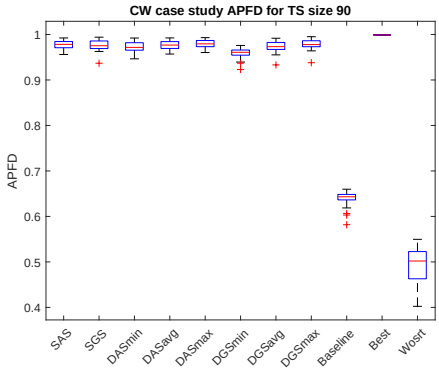
In Sections 7.4.1 and 7.4.1 we proposed two different test similarity measures (i.e., *GSbSim* and *GSbSim*) that have been used as test prioritization criteria

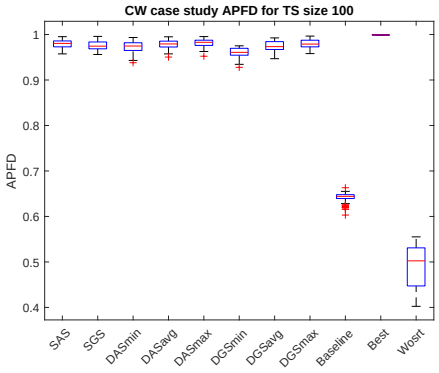

CW case study APFD for TS size 50
APFD
1
0.9
0.8
0.7
0.6
0.5
0.4
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt

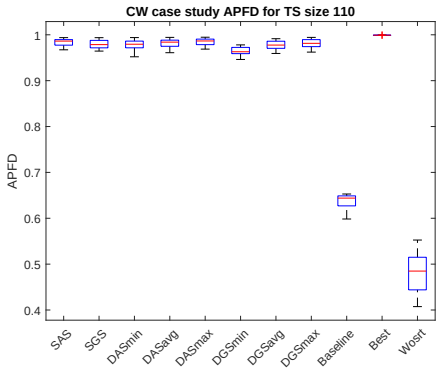

CW case study APFD for TS size 60
APFD
1
0.9
0.8
0.7
0.6
0.5
0.4
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt

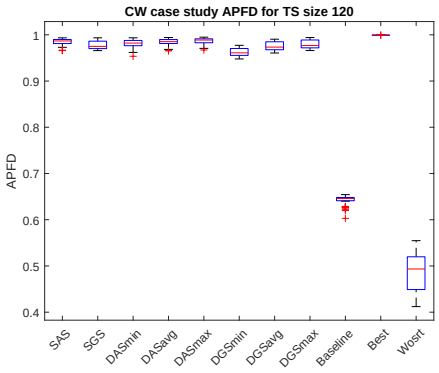

CW case study APFD for TS size 70
APFD
1
0.9
0.8
0.7
0.6
0.5
0.4
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


CW case study APFD for TS size 80
APFD
1
0.9
0.8
0.7
0.6
0.5
0.4
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


CW case study APFD for TS size 90
APFD
1
0.9
0.8
0.7
0.6
0.5
0.4
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


CW case study APFD for TS size 100
APFD
1
0.9
0.8
0.7
0.6
0.5
0.4
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


CW case study APFD for TS size 110
APFD
1
0.9
0.8
0.7
0.6
0.5
0.4
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt


CW case study APFD for TS size 120
APFD
1
0.9
0.8
0.7
0.6
0.5
0.4
SAS
SGS
DASmin
DASavg
DASmax
DGSmin
DGSavg
DGSmax
Baseline
Best
Wosrt

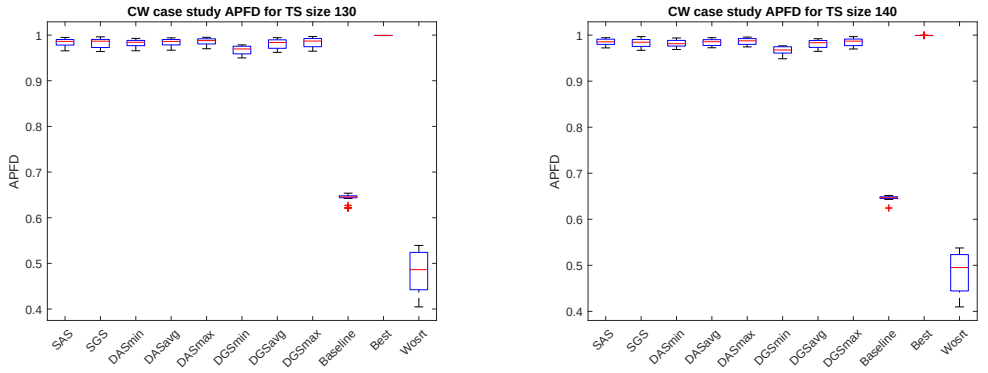


Figure 7.6: Distribution of test scenarios per technique and test suite for CW case study. Note that for the dynamic techniques results are shown as minimum, average and maximum to make the charts of a reasonable size.

Table 7.4: RQ2: APFD percentage comparison between the static prioritization algorithm variants (i.e., SGS and SAS) with respect to the dynamic ones (i.e., DGS and DAS).

A	B	IT			CW		
		A >B	A = B	A <B	A >B	A = B	A <B
SGS	DGS	19.2%	0.1%	**80.7%**	46.3%	0.4%	**53.3%**
SAS	DAS	20.8%	0.3%	**78.9%**	40.4%	0.3%	**59.3%**

both for the dynamic and the static test prioritization approaches. This was proposed for the context of configurable simulation models, and rely on the euclidean distance to measure the similarity of test cases. The third RQ aimed at evaluating which of the test prioritization criteria performed best for prioritizing test cases in the context of configurable simulation models. To make a fair comparison, we compared SGS and SAS (i.e., both static prioritization algorithms) with DGS and DAS (i.e., both dynamic prioritization algorithms). The reason is that they both have the same prioritization algorithm but different test prioritization criteria for evaluation.

Table 7.5 summarizes the obtained the results for RQ3, in a similar way as previous RQs. SAS and DAS outperformed SGS and DGS in both case studies, meaning that the distance measure *ASbSim* proposed in Section 7.4.1 performed better than *GSbSim* proposed in Section 7.4.1 in the context of configurable simulation models. For the IT case study, SAS outperformed SGS in 76.6% of test scenarios, whereas SGS outperformed SAS in 23.4%. When considering the dynamic test prioritization algorithms, DAS outperformed DGS

in 76.2% of test scenarios, whereas DGS outperformed DAS in 23.8%. In the CW case study, the results are tighter, SAS outperforming SGS in 57% of test scenarios and DAS outperforming DGS in 61.9% of test scenarios. Conversely, SGS outperformed SAS in 43% of test scenarios and DGS outperformed DAS in 38.1% of test scenarios.

When considering the APFD values, as what happened in RQ2, the average improvements were not high. SAS outperformed the SGS technique by 1.89 and 0.21% in the IT and CW respectively. As for the dynamic techniques, DAS outperformed the DGS by 1.72 and 0.36% in the IT and CW case studies respectively.

Table 7.5: RQ3: APFD percentage comparison between algorithm variants based on different test prioritization criteria.

A	B	IT			CW		
		A >B	A = B	A <B	A >B	A = B	A <B
SGS	SAS	23.4%	0%	**76.6%**	43.0%	0%	**57.0%**
DGS	DAS	23.8%	0%	**76.2%**	38.1%	0%	**61.9%**

RQ4 – Algorithms Configurations

The fourth RQ aimed at analyzing the effects different configuration settings of the dynamic test prioritization algorithm variants. To this end, we selected different configurations (as explained in Section 7.5.2). We evaluate how $start - upsize$ and $reallocationsize$ affected the performance of the dynamic algorithm for prioritizing test cases. Furthermore, we applied Spearman's rank correlation statistical tests to assess the correlation between APFD and mentioned selected configurations. On one hand, Tables A1 and A2, that can be found at the appendix, report the results of Spearman's rank correlation for both case studies when the start-up size varies. In this case, if ρ value is positive, it indicates that the performance of the approach increased with the number of test cases to obtain the start-up data, whereas a negative value indicates the opposite. The $p - value$ indicates that the performance increase/decrease has statistical significance if its value is lower than 0.05. On the other hand, Tables A3 and A4, that can be found at the appendix, report the results of Spearman's rank correlation for both case studies as long as the reallocation size of test cases varies. In the case of ρ value being positive, it indicates that the performance of the approach increased when the number of

test cases in the reallocation was higher, whereas a negative value indicates the opposite. Similarly to the previous tables, the $p - value$ determines the statistical significance (i.e., there is statistical significance if $p - value < 0.05$).

When varying the number of test cases to obtain the start-up data, it can be seen that the performance did not vary significantly for the DAS technique in any of the case studies for any case. Conversely, for the DGS technique, the number of test cases to obtain the initial start-up data did have a statistically significant impact on performance when considering the APFD in some cases. Regarding the IT case study, the performance showed a positive correlation with statistical significance when the start-up size increased and the reallocation size was 200. For the CW case study, the performance correlated positively when the reallocation size was 100 and 200, for all the cases, but also for some cases when the reallocation size was 50.

As for the number of test cases to be reallocated, this parameter did not have any relevant impact on any of the cases for the DAS technique in the IT case study, although it had some impact in a few cases for the CW case study. For instance, the reallocation size had a negative impact at the time of increasing the number of test cases to be reallocated when the test suite size had 100 or 110 test cases.

For the DGS technique, however, the results differed in both case studies. For the IT case study, the higher the number of tests to be reallocated, the better the performance, particularly when the start-up size was larger. Conversely, for the CW case study, the higher the number of test cases to be reallocated, the worse the performance of the algorithm, especially when the start-up size was below 10 test cases.

RQ5 – Upper and Lower Bound Comparison

RQ5 evaluated how the selected techniques performed when compared with the optimal and worst-case test prioritization for each scenario. The reason to compare results with respect to both best and worst case test prioritization scenarios is two-fold. On one side, we analyze how far from the optimal test prioritization the selected techniques performed in terms of fault detection. On the other side, the worst-case scenario ensures that obtained results are not biased. To this end, we compared the average values of the 500 scenarios for each of the test prioritization approaches as well as the optimal test scenarios. Figures 7.5 and 7.6 clearly indicate that our approaches are closer to the optimal test prioritization than to the worst-case one. Table 7.6 reports the

percentage of improvement of the optimal test prioritization with respect to the proposed approaches in terms of APFD. In the IT case study, the optimal test prioritization improved the static test prioritization approaches by 7.42% and 9.14%, and dynamic test prioritization was improved by 7.23% and 8.80%. The CW case study proved more promising for our techniques, as the prioritization improvement dropped to 1.97% and 2.18% for the static test prioritization approaches and to 2.07% and 2.42% for the dynamic test prioritization approaches.

Table 7.6: RQ5: Average percentage of improvement of the optimal test prioritization with respect to the proposed test prioritization algorithms in terms of APFD

Case Study	SAS	SGS	DAS	DGS	Baseline
Industrial tanks	7.42%	9.14%	7.23%	8.80%	29.01%
Car Windows	1.97%	2.18%	2.07%	2.42%	36.09%

When comparing results with the Lower Bound, Table 7.7 shows the average improvement extent of our algorithms with respect to the worst possible test prioritization in terms of average APFD values for all scenarios. On one hand, results for the static test prioritization approaches were 63.93% and 63.25% above worst case for the IT case study and 50.13% and 50.02% for the CW case study. On the other hand, results for the dynamic test prioritization algorithms showed that they improved the worst case test prioritization by 64.01% and 63.39% for the IT case study and by 50.07% and 49.80% for the CW case study.

Table 7.7: RQ5: Average percentage of improvement of the proposed test prioritization algorithms with respect to the worst test prioritization in terms of APFD.

Case Study	SAS	SGS	DAS	DGS	Baseline
Industrial tanks	63.93%	63.25%	64.01%	63.39%	52.96%
Car Windows	50.13%	50.02%	50.07%	49.80%	23.50%

7.5.4 Discussion

The first RQ compared the proposed approach with a baseline technique. Our approach significantly outperformed the baseline technique for all created test

scenarios in both case studies. Two main conclusions are obtained from these results: firstly, the proposed problem (i.e., test prioritization in the context of product lines) is a non-trivial task. This has also been found in other test prioritization studies on product line engineering [AWSE19, WBA+14], although in this context, the problem is different. The second conclusion is that the proposed approach as well as the test prioritization criteria for the context of configurable simulation models are performing well. Thus, we can answer the first RQ as follows:

> *The proposed test prioritization algorithms perform better than the selected baseline test prioritization technique, meaning that they work properly in this context and are recommended for use by practitioners.*

The second RQ compared the static test prioritization algorithms with the dynamic ones, which is one of the major contributions of this chapter. For both case studies, generally, the dynamic test prioritization, which prioritizes test cases dynamically by considering their test results, performed better than more traditionally employed static test prioritization techniques. Note that the dynamic test prioritization approach is general for any product line, although we adapted it to the context of configurable simulation models. A conclusion obtained from this RQ is that, at least for the context of configurable simulation models, the dynamic test prioritization algorithms perform correctly, meaning that using the test verdicts obtained during the execution of the test cases for re-updating the test orders can indeed help improving the fault detection rate. We can thus answer the second RQ as follows:

> *Overall, the dynamic test prioritization algorithms performed better than the static test prioritization algorithms.*

The third RQ compared the two test prioritization criteria proposed for the context of test prioritization of configurable simulation models. In this case, the test prioritization criterion that considers all signals together (i.e., the test prioritization criterion based on All-Signals Similarity, denoted as AS) for measuring the distance among test cases performed better than the test prioritization criterion which groups the signals according to the three different groups of signals described in Section 7.4.1 (i.e., the test prioritization criterion based on Grouped-Signals Similarity, denoted as GS). An explanation for this might be the way the similarity is measured. In the case of the *ASbSim*

similarity measure, the distance between all signals is considered between two products, no matter whether these products have these inputs or not. Conversely, in the case of the *GSbSim* similarity measure, when a product has a signal and the other one doesn't, the maximum distance is applied (i.e., 1). On prioritizing test cases, thus, when a product with many signals is selected, the following product will be a product with few signals. Usually, products with few signals are simple products, which subsequently have low probability of containing faults. Consequently, when considering this similarity measure for the test prioritization criterion to prioritize tests, ranking is given to small products with low probability of containing faults. We can thus answer the third RQ as follows:

> *Overall, prioritizing test cases by considering the ASbSim similarity measure for the test prioritization criterion performs better than employing the GSbSim similarity measure.*

The fourth RQ measured how the configuration parameters for the dynamic test prioritization algorithm affected its performance. Especially for the DGS technique, when varying the start-up size, we noticed that the higher the number of tests to initialize the start-up value, the better the performance of the algorithm in terms of fault detection rate. A potential reason for this might be the conclusion obtained by [KIJT18] et al., where it is suggested that history-based test prioritization approaches require a long start-up process. With regards to the reallocation size, we noticed that its variation affected the performance of the DGS technique. Nevertheless, unlike with the start-up size parameter, results differed from one case study to another. For the IT case study, the higher the reallocation size, the better the performance of the technique in terms of APFD. There was specifically strong correlation when the start-up size was large. Conversely, for the CW case study, the higher the reallocation size, the lower the performance of the technique in terms of APFD. There was specifically a negative correlation in those cases where the start-up size was low. These results, again, lead us to conclude that the longer the start-up process for updating the historical database, the better the performance of test prioritization techniques. Additionally, the difference of results when varying the reallocation size could be driven by the nature of the tests which were generated for the case studies, rather than the techniques themselves. Bearing this in mind, we summarize the answer to the fourth RQ

as follows:

> *The configuration of the number of tests to reallocate and the start-up size has little effect on the DAS technique, but it does have effects on the DGS technique. The start-up size has a positive impact, especially when the number of tests to be reallocated is high (i.e., 100 or 200 test cases). The reallocation size can have a positive impact when the start-up size is large (i.e., 20 or 50), but it can indeed have a negative impact when said size is small (i.e., 1 to 10). Subsequently, the largest number of tests to initialize the historical database is recommended.*

The last RQ compared our algorithms with respect to the optimal and worst test prioritization cases for each of the test scenarios. As explained in the results section, our approaches were much closer to the optimal prioritization than to the worst. However, and especially for the first case study, the average improvement of the optimal test ordering with respect to our algorithms was between 7.42% and 9.14%, meaning that there is still room for improvement. This could be a potential avenue for future research. Subsequently, we can answer the last RQ as follows:

> *The selected techniques are quite close to the optimal test ordering, although there is still room for improvement, suggesting that further research might prove valuable. On the other hand, the proposed techniques improve the worst case test prioritization scenario by between around 50% and 64% - these values can be considered quite high - and thus, our techniques can start being used by practitioners.*

7.5.5 Threats to Validity

We now summarize the threats that our empirical evaluation is exposed to and how we tried mitigating those threats.

Internal validity

One internal validity threat that our evaluation is exposed to is the generated mutants. We generated 246 and 209 mutants for IT and CW case studies respectively, and we later applied a filter following different criteria. Although the initially generated number of mutants was not large, note that in the context of Simulink the models include a physical layer that is composed of

complex mathematical blocks. This makes the execution of test cases very time-consuming and thus, it is not feasible to use a large set of mutants. Note, however, that the amount of mutants is similar or larger than those used in other studies involving Simulink models [MNBB16, ASEZ16, AWSE19, MNBB19, MNBB15, HBT16, AWSE16a, LLNB17, LNLB19, LLN+16, LTMHT14]. Besides, we tried mitigating this threat by removing unnecessary mutants, such as duplicated mutants, as recommended in previous studies [PJHLT15].

External validity

In all empirical studies, an external validity threat is the number of case studies used to evaluate the results. In our case, we used two case studies. Note, however, that the employed case studies are from two different domains and have different characteristics. Furthermore, note that the size of our case studies is large (300 and 227 blocks); according to a previous study, more than half of the public models have less than 100 blocks and around 75% of the models had less than 300 blocks [CMM+18]. Thus, we can consider that the selected case studies are large when compared to most of the public models.

Conclusion validity

A conclusion validity threat in our evaluation refers to how good the tests might be at detecting faults and not. To reduce this threat we divided the evaluation in 10 different experimental scenarios of different test suite sizes (from 50 to 140). For each of these experimental scenarios, we randomly generated 50 different test suites by selected test cases from a pool of 150 test cases. This permitted us to have different test prioritizations with a wide and diverse range of test cases, reducing the conclusion validity threats.

7.6 Related Work

Test case prioritization has been widely investigated in the past, where several approaches have been proposed and empirically evaluated [RUCH99, RUCH01, ERP14, EMR02, EYHB15, HFM15, JC15, JH03, KTH05, KKT08, RBT13, ZHZ+13, HZZ+16, LMP16]. When focusing more on static and dynamic test prioritization approaches, Luo et al. [LMP16, LMZP19] compared the performance of test case prioritization methods from a static and a dynamic perspective for Java programs. Nevertheless, it is remarkable that the concept of static and dynamic test prioritization is different. In our context, the

term dynamic test prioritization refers to the methods that re-prioritize an initially prioritized test suite by making use of the test results, whereas [LMP16, LMZP19] refer to dynamic test prioritization to the regression techniques which employ the code coverage information of previous software version executions to re-prioritize test cases. Another remarkable aspect is that the Luo et al. proposal focuses on the study of individual software programs, while the approach we propose focuses on product lines. More similar to this study's approach is the one proposed by Pradhan et al. [PWA+18, PWA+19]. In their approach, they aim at re-prioritizing test cases based on rules inferred from historical data in context of continuous integration, showing improvement over traditional static test prioritization approaches. There are several key differences between their approach and ours. The first is that they require historical data to mine rules between test cases and failures, whereas our approach does not consider an initial historical database. The second key difference is that their approach is not intended for product line engineering, whereas our test prioritization algorithm is designed for configurable systems.

In the context of SPLs and configurable systems, testing has gained important attention over the last few years [LUV09, ER11, NdCMM+11, JHF11b, LKL12, dCMMCDA14, LHLE15, LKJ20]. Most of the studies focus on proposing novel approaches for generating relevant products that will ensure finding as many faults as possible due to interaction of features [HPP+14, HPHT15, PSK+10, POS+12, CDS08]. In the context of our study, these techniques are used to generate the initial products, but we do not propose new approaches for sampling new products. Thus, we focus on optimizing the testing process by considering already generated test cases and products.

Other approaches include test optimization techniques for the context of product line engineering, including test suite minimization [WAG13, WAG15], test case selection [WGAL13, WAGL16a, AWSE16a] and test prioritization [AHTM+14, SSRC14a, PSS+16, DPC+15, WBA+14]. As for test prioritization approaches, two main dimensions can be differentiated: (1) test prioritization at the Domain level and (2) test prioritization at the Application level. The former dimension refers to prioritizing the order at which the products will be tested, but do not consider which order the test cases will follow at each product. Among these approaches, the community has considered product prioritization based on statistical techniques [DPC+14, DPC+15], based on similarity of products [AHTM+14, AHTL+16], search techniques [PSS+16] and based on other criteria such as product complexity or number of changes in the

assets of specific features [SSRC14a, SSPRC15]. The latter dimension refers to prioritizing test cases for optimally testing a specific product once its order has been defined. To this end, several approaches have been proposed, including search techniques along with specific objective functions [WBA+14, AWSE16b, AWSE19] or delta-oriented approaches for integration testing [LLL+15].

Our approach is different from all these techniques in several aspects. Firstly, our approach considers both levels, the domain engineering level (i.e., we prioritize products of configurable systems) as well as the application engineering level (i.e., we consider the tests that will be applied to test a specific product) by giving the option of prioritizing a product with an associated test case and later executing the test of another product. Additionally, to the best of our knowledge, there are no approaches in the context of product line engineering that consider the test execution result to re-prioritize the order in which tests are executed, something that we propose as one major contribution of this chapter.

The approach for dynamic test prioritization for configurable systems has been adapted to the context of configurable simulation models. In this context, there have been several approaches that have aimed at proposing scalable solutions for the area of testing. Matinnejad et al. proposed a test generation approach for Simulink models [MNBB16]. In their extended version, they also proposed a test prioritization technique for the context of simulation models [MNBB19]. Our previous work focused on black-box techniques for regression test selection of simulation models [AWA+18, AWM+19], where we adapted the Euclidean distance for measuring the similarity among test cases, which is used in the approach presented in this work. Notice, however, that all these approaches focus on simulation models that are not configurable, whereas in this research work we propose an approach for configurable simulation models.

7.7 Conclusions

In this chapter we propose a dynamic test prioritization approach for software product lines, which, for a given statically prioritized test set, dynamically updates the test case order, leveraging results information of test being executed. To this end, two algorithms (i.e., static and dynamic) have been proposed and different configurations of the proposed dynamic algorithm parameters have been analyzed. Additionally, the approach has been adapted to the context of configurable simulation models, for which two test prioritization criteria have been proposed to measure test similarity. In total, by combining the static

and dynamic test prioritization algorithms with test prioritization criteria, four algorithm variants are derived and analyzed. An empirical evaluation with two case studies showed that the proposed approach performed better than the selected baseline technique in all cases. Furthermore, between 53.3% and 80.7% of the times the dynamic algorithm performed better than the static algorithm. When analyzing the performance of the proposed test prioritization criteria, the one that employed the information from all the signals together in the model obtained better performance (i.e., AS performed better by between 57.0% and 76.2% of the times). The analysis of the $nStartUp$ configuration parameter of the dynamic algorithm reflected that the higher the number of tests executed to feed the test history, the better the performance of the dynamic algorithm. On the contrary, the analysis of the configuration parameter $nRealloc$ seemed to be dependent on each case study. Finally, the results obtained by the proposed algorithm variants were close to the optimal ordering of the tests and outperformed the worst case test prioritization scenario by 50% up to 64%. Overall, the proposed dynamic prioritization approach performed better than static prioritization in the assessed context, and it is therefore recommended to be used by practitioners.

Part III

Final Remarks

Chapter 8

Conclusion

Contents

This is the final chapter of the thesis and it is structured into two main sections. Section 8.1 presents the conclusions of the dissertation. For this purpose, the contributions are summarized, the validation of the hypotheses is analyzed and the main limitations are discussed to conclude with the lessons learned. Section 8.2 describes the short and mid-term future work to continue the research of this dissertation and its transfer to industry.

8.1 Conclusions

8.1.1 Summary of the Contributions

CPSs are one of the core-enabling technologies of the industry of the future, which requires high flexibility to be competitive. By providing high configurability, we address flexibility while adding more complexity to demanding systems. Optimization of HCCPS testing is a paramount concern due to the demanding changes in the market and the need for rapid response. In this thesis we propose methods that address this issue in a cost-effective manner.

The first and main contribution of this thesis consists of a dynamic testing prioritization approach. The proposed method establishes a (static) test plan that executes pairs (of product and test cases) in an iterative manner, reordering dynamically the test plan after every iteration based on previous executions. The contribution is generally intended for use in any kind of product lines. Given that HCCPS testing is more time-consuming and costly, the potential benefit is even greater. The empirical evaluation conducted with two case studies suggested that the proposed dynamic approach performed better than the selected baseline technique.

A secondary set of contributions is also presented, which has contributed during the exploratory stage to the definition of the main contribution.

The second contribution (first contribution during the exploratory stage) proposed a method to explore the selection and prioritization of a small group of test cases for previously prioritized products in an iterative manner. To this end, four variants of weight-based genetic algorithms were employed. The approach was empirically evaluated with the UAV case study and the results showed that the proposed algorithms improved the baseline. In addition, the results allowed concluding that the black-box test case appearance frequency metric, which enhances dissimilarity of test cases, improves the fault detection.

The third contribution (second contribution during the exploratory stage) proposed the comparison of test case selection methods based on structural coverage. Specifically two metrics for measuring the structural coverage at single product or product-line level were employed in combination with three coverage criteria (i.e. DC, CC and MC/DC). The method was empirically evaluated with the CW case study and results suggested that the Domain level method that considers the overall coverage of the product line obtains better results. Additionally, the pair concept of selecting a test case with a product was envisioned, establishing another basis for the main contribution.

8.1.2 Hypothesis validation

In this thesis, three hypotheses have been raised in Section 4.2. This section analyzes each of the contributions and argues whether the stated hypotheses have been validated.

First hypothesis

The first hypothesis is stated as follows:

> Combining Domain-level information with Application-level information using search-based techniques provides an iterative selection of test cases of a HCCPS that improves fault detection over traditional techniques.

To evaluate this hypothesis we proposed an approach that allocates small groups of test cases to previously selected and prioritized products. In a first stage we rely on well-established techniques for product selection and prioritization (i.e., we select a representative set of products for a pairwise coverage with the ICPL algorithm [JHF12] and prioritize the products using the VC&CC criterion [SSRC14a]).

At a second stage, our approach proposes a novel algorithm that allocates small groups of test cases to the pre-selected and pre-prioritized products using search-based techniques. To this end, a fitness function was defined at two levels (i.e., at the Application and at the Domain level) to guide the search taking advantage of weight-based search algorithms. At the Application level we employed Fault Detection Capability (FDC), Test Execution Time (TET) and Test Case Appearence Frequency (TCAF) measures to assess the effectiveness of small groups of test cases allocated to specific products. At the Domain level we aggregate Application-level fitness function measurement of each product,

weighted with the priority of each product according to the pre-calculated product prioritization.

Two different weight-based search algorithms were employed (i.e., Weight-Based Genetic Algorithm (WBGA) and Random-Weighted Genetic Algorithm (RWGA)). Furthermore, two different alternatives of fitness functions were designed (i.e., one considering FDC and TET measures, and the second one considering the two mentioned above, and TCAF). In total, combining algorithms and fitness alternatives, four search algorithm variants were defined. In addition, the selected UAV case study was evaluated with 24 artificial problems whereas Random Search (RS) and *Traditional test approach* were taken as baselines.

The main drawback of this approach is that it requires an initial FDC history in order to run the algorithm. In our case we solved this matter carrying out a series of previous training runs. However, it must be taken into account that these training runs place an added workload on engineers (i.e., in the case where no FDC history is available it would be necessary to build a training set of test cases and mutants to feed required data).

Results from the empirical evaluation showed that the proposed search algorithms outperformed both baselines. Moreover, the RWGA algorithms in combination with appearance frequency objectives obtained the best performance when detecting faults. Regarding the size of the test suite, we found that the bigger the test suite the better the performance.

Considering the obtained results, we concluded that the proposed approach that combines Domain level information with Application level information outperformed the traditional techniques, in the evaluated case study. Consequently, it can be assumed that the stated first hypothesis has been validated.

Second hypothesis

The second hypothesis is stated as follows:

> Considering the structural coverage information of a HCCPS at Domain level helps optimizing fault detection results for time-budget constrained scenarios.

To evaluate this hypothesis we proposed three different test case selection methods based on structural coverage information considering a given time budget. The proposed test case selection methods employ the structural

coverage metric at two different levels: (i) the traditional use of structural coverage for individual products, named Product Structural Coverage (PSC), and the structural coverage that considers the entire product line, named Product-Line Structural Coverage (PLSC). Moreover, the three test case selection methods were analyzed with three white-box coverage criteria (i.e., DC, CC and MC/DC). Two out of the three selection methods require an initial step of product selection and prioritization, for which well-known techniques were employed. Specifically, product selection was performed for pairwise coverage with the ICPL algorithm and product prioritization was performed using the VC&CC criterion.

The Application level Method (AM) seeks to maximize the PSC metric. To this end, the developed algorithm selects test cases in order to thoroughly test prioritized products one after another until the time-budget is consumed. The Domain level Method (DM) aims at maximizing the PLSC metric. For this purpose, the developed algorithm selects each time such test case that most increases the coverage of the entire product-line (i.e., the PLSC metric, without product prioritization restriction). In the Combined Method (CM), the two previous methods were combined. Products are prioritized, but the metric employed to select the test cases is PLSC. Thus, the developed algorithms seek to maximize the coverage of the entire product-line by respecting the pre-established prioritization product order.

We evaluated the three selection methods with the CW case study. In addition, 25 artificial problems were selected. The traditional testing approach was employed as baseline.

The empirical evaluation conducted with the CW case study provided results to state that overall, the proposed methods outperformed the baseline. In addition, results suggested that considering coverage information at the domain engineering level helps detecting more faults, particularly when time budgets are low. Regarding the three coverage criteria (i.e., DC, CC, MC/DC), it can be concluded that all three perform very similarly in the case study. Considering the empirical evaluation, we conclude that the stated second hypothesis has been validated.

Third hypothesis

The third hypothesis is stated as follows:

> Considering the results of tests executed on products allows a dynamic prioritization of tests of a SPL that improves the capability to detect faults.

To evaluate this hypothesis, we proposed a dynamic test prioritization approach and empirically evaluated this with two case studies (i.e. CW and IT case studies).

The approach consists of three stages: firstly, valid pairs of products and test cases are formed. To this end, we formalized the *valid pairs* concept. Secondly, the given pairs are statically prioritized. The prioritization employs well-known product and test case similarity-based metrics. However, with the static prioritization algorithm we introduced a weight-based criterion that combines similarity metrics of both Domain and Application levels to prioritize pairs. In the third stage, our novel algorithm performs the dynamic prioritization of pairs. For this purpose, product fault-detection-capability information obtained from pairs being executed is leveraged to re-organize pairs.

The approach was adapted to the context of configurable simulation models. When adapting the approach to measure test case similarity, after studying the different type of groups of signals, two test prioritization criteria were proposed (i.e., named All-Signals Similarity of pairs (AS) and Grouped-Signals Similarity of pairs (GS)). By the combination of the two algorithms (i.e., the static and dynamic pair prioritization) and the two proposed prioritization criteria, in total, four algorithm variants were derived.

Two case studies (i.e., CW and IT) were selected in order to empirically evaluate the approach. In addition, different experimental scenarios were generated by combining 10 test suite sizes with 50 different test case selections. In total, 500 experimental scenarios were generated per case study. In relation to the 2 configuration parameters of the proposed dynamic algorithm, we defined: (i) 5 different sizes of test suites for the warm-up phase, and (ii) 6 different sizes to be taken into account when selecting a subset of test cases to be dynamically reorganized. Thus the dynamic algorithm was run with 30 different configurations for each experimental scenario. A test prioritization technique that minimizes input signals diversity was employed as baseline.

Before proceeding with the results, it is important to note that the evaluation was carried out exclusively for the adaptation made to the configurable simulation models.

Results from the empirical evaluation showed that the proposed approach performed better than the selected baseline technique in all cases. In addition, the dynamic algorithm performed better compared to the static algorithm in general. The results of comparing prioritization criteria based on groups of signals indicate that the use of All-Signals performs best. The analysis of the warming-up configuration parameter (i.e., $nStartUp$) of the dynamic algorithm reflected that the higher the number of tests executed to feed the test history, the better the performance of the dynamic algorithm. Considering the results of the empirical evaluation, we conclude that the third hypothesis has been validated.

8.1.3 Limitations

This section discusses the main limitations of the proposed methods. These limitations are directly related to the number of case studies and the selected tools.

During the experimentation stage, the approaches have been evaluated with a single case study, while the evaluation of the main contribution has been carried out with two case studies. We consider that in order to generalize the results it is convenient to complete the evaluation with other case studies. Nevertheless, we have selected and improved the case studies to ensure that they are of adequate complexity and size and that they are representative of the problem. Given the difficulty of incorporating new case studies, we have additionally used the artificial problem technique, which allowed us to analyze different characterizations of each case study multiple times with the same case.

As described in Section 4.4.1, the experimentation of this thesis has been carried out based on three tools: (i) FeatureIDE has been employed to model the variability of HCCPSs and to generate the product configurations. (ii) Simulink has been used for the modeling and simulation of the HCCPSs and (iii) MATLAB has been used to implement the algorithms of the proposed methods and to automate the creation, execution and evaluation of the case studies.

FeatureIDE has been selected since it is an open source tool that provides all the functionalities required for this research (i.e., variability modeling and valid product configuration generation). However, if necessary, this tool can be replaced with another which provides the required functionalities [AOCN20, HPF19].

For the evaluation of the approach we have employed simulation models of HCCPS systems based on dynamic behavior, i.e., systems with time-varying behavior. We have selected Simulink for the modeling and simulation of HCCPSs as it is a tool with proven credibility in both academia and industry [MNBB19]. A certain type of blocks (e.g., Constant, Gain, Logic, MultiplePortSwitch, RelationalOperator, Saturate, Sum, Switch, StateMachineCharts, etc.) have been used during the design of the models in Simulink. Taking into account the wide catalog of blocks that the tool offers, it might be convenient to perform additional tests with other blocks in order to extend the scope of the proposal.

MATLAB provides excellent integration with Simulink, and this is the main reason why we have employed it for the development of the algorithms and the automation of the creation, execution and evaluation of the case studies.

Although we consider this type of system to be representative, in order to validate the proposal with other types of designs and blocks it is necessary to carry out new evaluations and adapt the developments. The modeling of new simulation models may be expensive, however, adapting the developments for the automation of the generation of products, test cases and mutants, is not.

To extend the generalization, it is be convenient to validate the proposal with other simulation tools outside the MATLAB/Simulink boundaries, with solutions such as Dymola, BCVTB, Gazebo, ISAAC-Nvidia, etc. Nevertheless, this would mean rebuilding both the HCCPSs models and the developments for handling them, thus requiring considerable effort. We understand that the rebuilding of the developments would make sense if the proposal was applied to a specific context of a company that already has tools for simulation and is considering applying the approach.

In addition, opening up to new tools [ZSV18] would make it possible to extend the proposal to other concepts that allow its validation in more complex scenarios. For instance, some of these tools (e.g., Dymola, BCVTB) also allow advancing to Co-Simulation scenarios, allowing the integration of different simulation tools [SAM+17, ELM+17]. Other tools (e.g., GAZEBO) enable development in open source contexts [PUS17] or provide capabilities for learning reinforcement and photo-realistic simulations (e.g., ISAAC-Nvidia).

8.1.4 Lessons Learned and Conclusions

This section presents the most remarkable lessons learned from the research work conducted in this thesis.

- **We have learned the value of evaluating different metrics to select those that provide the best performance for the approach.** When considering optimization approaches, metrics are fundamental to obtain the best performance for the approach to be proposed. For instance, in [HPH+16], relevant test prioritization approaches were exhaustively evaluated for (software-based) regression testing. In this thesis, we have analyzed some of the metrics available in the field and we have also had to adapt and propose new ones.[1] Fundamentally, we have analyzed both white-box and black-box metrics but also metrics oriented towards measuring at the specific product level (i.e., at the Application level) as well as at the product-line level (i.e., at the Domain level). Experimenting with different metrics has allowed us to select the most appropriate ones for the proposed approach. Therefore, we recommend the community to evaluate metrics in future research.

- **The test optimization of products and test cases in a combined way (i.e., pairs) is a promising field.** In this thesis we have introduced the concept of "valid pair" and proposed a dynamic prioritization approach to optimize fault detection based on the aforementioned pairs. We believe that based on this concept and in combination with cost-effective metrics, it can be useful for solving other complex test optimization problems (e.g., test case generation or selection, etc.).

- **Simulation-based testing of HCCPS is effective at early stages.** There are multiple benefits of using simulation models for HCCPS testing (i.e., they allow the automation of both testing and evaluation, they enable the possibility of testing scenarios that may be dangerous, costly or even impossible to reproduce using a real prototype, etc.) [BNSB16, AWSE19]. However, we consider that simulation models provide a major advantage: the possibility to perform an early testing of the HCCPS. This technique enables the validation (up to a certain level) of the HCCPS cost-effectively, if we compare it with the required time and cost when using real prototyping. Furthermore, when this technique is applied to early HCCPS testing in the

[1] In [MAES19b] we published the set of metrics analyzed.

context of SAS testing strategy, as proposed in this thesis, is convenient for testing sampled products at Domain which further increases the benefit. Therefore, in HCCPS contexts, we recommend the use of simulation models for early-testing purposes.

- **Mutation testing is effective in the absence of real faults:** Conducting research experiments requires the availability of case studies to test the validity of the proposed approaches. In our case, since we did not have real faults to perform the evaluation, we relied on the Mutation-Testing technique as it has proven to be an effective substitute [PKZ+19, JH10]. The mutation testing technique has been adopted and adapted to the characteristics of the simulation model-based CPS testing domain [HBT16], as well as to the product-line domain [LS14, KMTG18]. In our context for HCCPS it is essential to design the mutant injection, ensuring mutant diversity (i.e., mutant operators of different types). In addition, a comprehensive distribution of mutants throughout the model must be ensured to have a proper coverage of features and feature interactions. Finally, these design guidelines must be complemented with a mutant selection process that ensures a clean-up of those mutants that are undetectable, duplicated or always detectable. These guidelines applied in the thesis have allowed us to evaluate the proposed methods. Hence, we recommend researchers and practitioners in the industry to employ the technique.

- **Automated tools for test case generation at early stages are effective:** Test case generation is properly a research area where a large body of research has been conducted [ABC+13, ABHPW10, PSK+10, EBG12, AWM+17b]. However, in this thesis, we have proposed an approach closer to the way industrial companies design and simulate CPSs. Specifically, we have relied on three test case generation techniques: (i) Domain knowledge from engineers, (ii) Tool-assisted test case generation for the MC/DC coverage criterion, and (iii) Tool-assisted random generation of valid test cases. Test cases generated by domain experts on the basis of domain knowledge are traditionally costly to develop, but effective. However, taking into account that the focus of the thesis lies on optimizing early-stage HCCPS testing, we have also relied on tool-based techniques. From this experience we conclude that the tools are perfectly valid for the given context. However, these approaches could be enhanced by applying techniques (e.g., search-based techniques) described in the literature [AWM+17a], which on the other hand

might not be easily accessible for many industrial companies.

- **The research methodology employed has resulted effective, but other variants should be studied for future research.** We have been inspired by the Design Science Research (DSR) methodology variant proposed by Vaishnavi [VK04], which has proved to be successful for the iterative and exploratory approach that has been proposed. However, considering that DSR has evolved and that there are more complete proposals, we believe that for future work it would be desirable to analyze other methodology variants that can provide additional benefits (e.g., that may allow us to further explore both the problematic and the generalization).

8.2 Perspectives and Future Work

In this section we present the next steps of the thesis. Firstly, the transfer of the contributions to the industry are discussed. Secondly, the identified research lines to extend the contribution are presented.

8.2.1 Industry Transfer

This thesis has been developed in the Software and System Engineering research group belonging to the engineering faculty of Mondragon Unibertsitea. The faculty has a long tradition of collaboration with the industry through research and transfer projects. We have identified three companies that might be interested in our proposed methods.

- **Alerion Technologies** is a leading company that develops solutions for the autonomous flight of UAVs for infrastructure inspection. The quality assurance process they perform requires four testing phases (unit testing, integration testing, simulation-based system testing and field tests). Although they do not work with product lines, their CPS systems work with numerous parameters that must be taken into account when testing. In the context of the TESTOMAT project we have collaborated to instrumentalize the simulation-based testing phase for its automation, based on Docker containers and GAZEBO simulation technology.[2] In addition we have developed a tool for the selection and prioritization of test cases based on the engineers' domain knowledge. We believe that the methods proposed in this thesis, adapted to their parameterized systems environment, could improve the early validation of Alerion's product.

- **CAF Power & Automation** is a railway company that provides design, development and maintenance services for electric traction systems, energy storage systems, and control and communication systems of rail vehicles. The company already manages the variability of multiple CPSs of a large product portfolio (different models of trains, trams, etc.). We consider that the adaptation of the methods proposed in this thesis could contribute to achieve the next level of efficiency in the quality assurance process of its portfolio.

[2]GAZEBO is an open-source robot simulation framework, supported by the Open Source Robotic Foundation. http://gazebosim.org/

- **ALDAKIN** is an engineering company specialized in automation solutions for productive processes, including human-robot collaboration contexts. We are currently collaborating with ALDAKIN in the VALU3S research project.[3] In this project, our faculty is working on a use-case with ALDAKIN: "Human-Robot Collaboration in a disassembly process with workers with disabilities". The university is working on “Simulation-based testing for human-robot collaboration” and “Test optimization for simulation-based testing of automated systems" topics in the project. For the test optimization part, the methods developed in this thesis could be adapted. However, as ALDAKIN does not work with product lines, it would be necessary to adapt the approach to a parameter-based system. In addition, the company employs other simulation tools (i.e., GAZEBO and ISAAC-Nvidia) that would also require development.

8.2.2 Further Research

Further research as well as new developments can be performed to complement this work. We propose lines of work aimed at reinforcing the validation of the results through new case studies and the validation of the approach for software-only systems.

- **Reinforce validation:** We have employed three case studies of significant size and complexity that we believe are representative for the problem. We have had to limit the number of case studies, as they are very expensive to develop and adapt, as well as to run the experiments. However, we believe that reinforcing the evaluation with *new case studies* would strengthen the validation. Specifically, we would like to incorporate a real case study, since in a real case, we could have not only access to the system itself, but also to the information from real data (test cases, failures, etc.).

- **Validate on software:** The evaluation of the proposed approach has been performed for HCCPS, since the need for optimization is particularly high due to the time-consuming testing processes. However, the approach is defined generically for product lines. Therefore, in order to extend the conclusions to other types of product lines (e.g., software-only-based SPLs), it would be necessary to adapt the approach and perform the corresponding empirical evaluation.

[3]VALU3S: Verification and Validation of Automated Systems' Safety and Security" is a project funded by ECSEL Joint Undertaking (JU) under grant agreement No 876852.

In order to implement the industrial transfer, we have identified several lines of work:

- **Tool Support:** The preparation and adaptation of the case studies for the thesis required the development of tools that allowed the creation and execution of the different artifacts in a semi-automatic way. We would like to complete the work further on different lines: (i) unifying the created different tools and (ii) automating and facilitating the deployment of the unified tool. In the first case, separate tools have been generated for creating and deriving the model-based products, test suites and mutants; as well as for managing the simulation executions and gathering results. Integration of those tools will allow a more efficient and simplified handling for industry adoption. In the second case, thanks to the advances that the MATLAB/Simulink toolset provides with container support, we plan to publish a container that allows to deploy the above-mentioned integrated system within a few minutes. We are confident that these two steps would provide agility in future experiments and facilitate its adoption by interested practitioners of the community.

- **Extend to parameter-based systems:** The industrial companies we collaborate with rarely develop product lines. However, we have identified that it is most common for them to provide configurability to their systems based on parameters. The above-mentioned companies (Section 8.2.1) may serve as examples. In order to transfer our approach to those companies, we consider it essential to analyze the necessary modifications to be made to our approach (e.g., solving how product similarity will be measured not having FMs.) in order to meet the requirements of configurable systems by parameterization.

- **Adapt to other simulation platforms:** The approach proposed in this thesis has been developed with the MATLAB/Simulink tool set. However, as we have indicated in the limitations section, other tools (e.g., GAZEBO, Dymola, etc.) are also used in the industrial field. Therefore, another line of development of the thesis consists of adapting the proposed approach to other simulation platforms. In this sense, we identify different avenues of development, which can be complementary among them: (i) The integration of the proposed algorithms with required third party tools, (ii) the re-implementation of the tools to manage the product-line artifacts (i.e., products, test cases, mutants) that in our case are model-based and simulation-tool dependent, (iii) the re-implementation of the tools to manage

the simulation execution and evaluation, (iv) the separation into different services (e.g., algorithms, artifacts and simulation control and evaluation) that allow us to integrate them into co-simulation contexts. These lines of work involve significant effort and resources, which we believe is feasible in a context of collaboration for industrial transfer.

We are aware that there are still aspects of variability to be investigated and we believe that extending the catalog of quality metrics and broadening the testing strategy will help us to establish the next steps.

- **Extend to new metrics:** Test quality metrics are essential to support verification and validation activities, yet this is even more necessary in the context of configurable simulation models since testing those models is expensive. We have collected, adapted and proposed a series of metrics in a previous work [MAES19b]. However, further research is needed in this field to respond to new needs as new aspects of variability specific to HCCPS [KNK+17] are incorporated into the research. A paper has recently been published [SLY+21] that defines a conceptual framework for the automatic configuration of CPSs. The paper defines a detailed classification of variation point types, which can be considered as an input to define new quality metrics related to variability aspects. We believe that the intrinsic characteristics of the HCCPS pointed out in these new aspects of variability will provide key information to measure the quality of the systems in a complementary way.

- **Extend to new strategies:** This thesis focuses on the Sample Application Strategy (SAS) for testing product-lines described by Pohl et al. [PBL05]. We specifically focus on the testing of the sampled products at the Domain level, but we don't propose an approach for the individual product testing at the Application level. We accommodated the SAS strategy in the context of HCCPS, specifically for testing configurable simulation models. We consider this scenario is valuable for early validation. However, the life cycle of HCCPSs encompasses other stages that are beyond the scope of this thesis. In this sense, Pohl et al. point to combined scenarios where both the benefits of early validation (SAS) and the benefits of maximum reusability raised in the Commonality and Reuse Strategy (CRS) strategy can be obtained. In addition, by opening the study to new strategies, we also consider it important to incorporate capabilities for incremental testing using regression scenarios (i.e., test one product, and apply regression techniques to test

following ones). We propose to examine the new context that the above-mentioned strategies imply in relation to the methods proposed. Possible future lines of research to be conducted in relation to the extension of the testing strategies: (i) complement the method with a test selection and prioritization approach for testing specific products at Application level (with information of what has been tested at Domain engineering using sampling), (ii) adapt the method for testing core features in the context of the combined CRS/SAS strategy and (iii) propose a method for incremental testing.

Appendices

Appendix A

Structural Coverage Introduction

Several approaches have been employed in the literature to measure the quality of a test suite (e.g., mutation testing [HPP+13a], structural coverage [SMP10], requirements coverage [RWSH08]). Among different types of coverage, Structural Coverage (SC) is a quality measure that has been in use since the first software testing techniques were referenced [MM63] in the 1960s. In the last decades, this measure has also been adopted by model-based engineering [UPL12]. The objective of the structural coverage measure is to determine the amount of code (or model) that has been exercised during a testing activity. Recent studies confirm the importance and topicality of the subject in the field [LKJ20].

There are multiple control-flow criteria for structural coverage [AO08]. Three of the most common criteria are: DC, CC and MC/DC [UL10]. The DC criterion checks the outcome of a decision. To this end, a minimum of two test cases are generated: one for a true outcome of the decision and another one for a false one. The CC criterion focuses on all possible values that every condition of the decision can take, irrespectively of the decision outcome. The MC/DC criterion examines that (i) every possible outcome for each decision is checked, (ii) every possible outcome for each condition is checked and (iii) each condition in a decision is shown to independently affect the outcome of the decision [HVCR01].

Consider as an example the code snippet in Listing A.1. The aforementioned coverage measures are obtained in the example with the test cases described in Table A1.

Listing A.1: Basic code example.

```
1       bool myFun (int x, int y)
2       {
3               bool result = false;
4               if (x>0 || y>0)
5               result = true;
6               return result;
7       }
```

Table A1: Test cases employed to obtain coverage criteria of the Listing A.1 example.

Coverage criterion	**Test Case**	**input1 x**	**input2 y**	**condition1 x>0**	**condition2 y>0**	**decision1 (x>0 || y>0)**
DC	TC_1	1	0	True	False	True
	TC_2	0	0	False	False	False
CC	TC_3	2	0	True	False	True
	TC_4	0	3	False	True	True
MC/DC	TC_5	4	0	True	False	True
	TC_6	0	5	False	True	True
	TC_7	0	0	False	False	False

The function `myFun` contains a decision (i.e., `(x>0 || y>0)` in column *decision1* in Table A1), which is composed of two conditions (i.e., `x>0` in *condition1* column and `y>0` in *condition2* column). A test case in this context is the input values x and y of the function, which allows for its execution. DC can be obtained for the example of Listing A.1 with two test cases: the first test case TC_1 has as inputs *x=1* and *y=0* and generates a *True* and *False* outcome to *condition1* and *condition2* respectively, whereas *decision1* obtains *True* value as outcome. The second test case has as inputs *x=0* and *y=0* and generates *False* outcome in both conditions and a *False* value in *decision1*. As both possible outcomes for *decision1* are obtained (i.e., *True* and *False* values in *decision1* column), DC is fully achieved. CC can be obtained with test cases TC_3 and TC_4, that provide all possible values (i.e., *True* and *False*) at every condition. MC/DC can be obtained with TC_5, TC_6 and TC_7 test cases. Every possible outcome for *decision1* is checked (i.e., TC_6 and TC_7 provide both *True* and *False* outcome in column *decision1*). Every possible outcome for *condition1* and *condition2* is checked (i.e., TC_5 and TC_6 provide both *True* and *False* outcome to the conditions). And finally, each condition of *decision1* independently affects the outcome of *decision1*. For example, modifying *input1* of TC_7 from 0 to 6 value shows how *condition1* affects to *decision1* outcome.

The coverage criteria can be measured in terms of test requirements. According to the definition provided by Ammann et al. [AO08] a *"Test Requirement (TR) is a specific element of a software artifact that a test case must satisfy or cover"*. For instance, if the goal is to obtain the DC of the example in Table A1, each test case attempts to satisfy a test requirement (i.e., TC_1 the *True* outcome of *decision1*, and TC_2 the *False* outcome).

Appendix B

Dynamic Prioritization Approach - Statistical Analysis supplementary tables

IT Case Study								
Alg	TS Size	Stat	Reallocation Size					
			5	10	20	50	100	200
DAS	50	ρ	-0.0023	-0.0007	-0.0066	0.0208	0.0517	0.0966
		Pval	0.9717	0.9911	0.9170	0.7440	0.4159	0.1276
	60	ρ	-0.0083	-0.0092	-0.0113	-0.0058	0.0114	0.0545
		Pval	0.8965	0.8848	0.8583	0.9273	0.8576	0.3912
	70	ρ	-0.0054	-0.0066	-0.0078	0.0136	0.0366	0.0569
		Pval	0.9327	0.9170	0.9019	0.8311	0.5641	0.3702
	80	ρ	-0.0063	-0.0060	-0.0068	-0.0034	0.0222	0.0698
		Pval	0.9212	0.9246	0.9143	0.9572	0.7264	0.2715
	90	ρ	-0.0019	-0.0037	-0.0039	0.0048	0.0193	0.0511
		Pval	0.9766	0.9540	0.9511	0.9400	0.7618	0.4209
	100	ρ	-0.0084	-0.0068	-0.0040	-0.0009	0.0152	0.0473
		Pval	0.8948	0.9150	0.9501	0.9889	0.8111	0.4563
	110	ρ	-0.0039	-0.0097	-0.0052	-0.0062	-0.0033	0.0201
		Pval	0.9513	0.8785	0.9346	0.9224	0.9592	0.7513
	120	ρ	0.0001	-0.0027	-0.0025	0.0023	0.0138	0.0431
		Pval	0.9983	0.9666	0.9680	0.9715	0.8279	0.4976
	130	ρ	-0.0049	-0.0033	-0.0037	-0.0081	0.0046	0.0239
		Pval	0.9386	0.9584	0.9538	0.8989	0.9418	0.7066
	140	ρ	-0.0075	-0.0087	-0.0129	-0.0111	-0.0029	0.0170
		Pval	0.9062	0.8911	0.8392	0.8620	0.9634	0.7885
DGS	50	ρ	0.0102	0.0077	-0.0045	0.0001	0.1318	0.1900
		Pval	0.8724	0.9036	0.9430	0.9990	**0.0373**	**0.0026**
	60	ρ	0.0102	0.0131	0.0093	-0.0120	0.0863	0.2120
		Pval	0.8729	0.8361	0.8836	0.8501	0.1739	**0.0007**
	70	ρ	0.0190	0.0161	0.0094	-0.0044	0.0849	0.2488
		Pval	0.7654	0.8002	0.8821	0.9454	0.1809	**0.0001**
	80	ρ	0.0103	0.0130	0.0089	0.0023	0.0561	0.2557
		Pval	0.8717	0.8380	0.8892	0.9710	0.3774	**0.0000**
	90	ρ	0.0273	0.0362	0.0343	0.0323	0.0628	0.3089
		Pval	0.6671	0.5694	0.5894	0.6117	0.3228	**0.0000**
	100	ρ	0.0220	0.0222	0.0184	0.0160	0.0246	0.2990
		Pval	0.7292	0.7271	0.7724	0.8016	0.6988	**0.0000**
	110	ρ	0.0296	0.0375	0.0327	0.0303	0.0295	0.2778
		Pval	0.6417	0.5552	0.6063	0.6331	0.6430	**0.0000**
	120	ρ	0.0280	0.0334	0.0317	0.0167	0.0009	0.2381
		Pval	0.6595	0.5988	0.6176	0.7930	0.9887	**0.0001**
	130	ρ	0.0163	0.0252	0.0236	0.0148	0.0087	0.2201
		Pval	0.7973	0.6922	0.7100	0.8162	0.8909	**0.0005**
	140	ρ	0.0267	0.0343	0.0330	0.0299	0.0048	0.1909
		Pval	0.6750	0.5894	0.6033	0.6375	0.9395	**0.0024**

Table A1: Correlation between start-up size and APFD based on Spearman's rank for IT case study.

CW Case Study								
Alg	TS Size	Stat	Reallocation Size					
			5	10	20	50	100	200
DAS	50	ρ	-0.0041	-0.0041	0.0035	0.0198	0.0270	0.0392
		Pval	0.9484	0.9484	0.9557	0.7555	0.6704	0.5373
	60	ρ	0.0020	0.0027	0.0220	0.0551	0.0858	0.1066
		Pval	0.9754	0.9656	0.7298	0.3859	0.1761	0.0926
	70	ρ	-0.0041	-0.0006	0.0110	0.0341	0.0633	0.0707
		Pval	0.9484	0.9926	0.8629	0.5915	0.3188	0.2651
	80	ρ	-0.0049	-0.0045	0.0024	0.0151	0.0286	0.0435
		Pval	0.9386	0.9435	0.9705	0.8123	0.6525	0.4935
	90	ρ	-0.0022	-0.0020	0.0010	0.0169	0.0270	0.0439
		Pval	0.9729	0.9754	0.9877	0.7909	0.6704	0.4896
	100	ρ	-0.0063	-0.0053	-0.0018	0.0024	0.0241	0.0568
		Pval	0.9214	0.9337	0.9779	0.9705	0.7045	0.3709
	110	ρ	-0.0035	-0.0082	-0.0043	0.0133	0.0404	0.0598
		Pval	0.9557	0.8970	0.9459	0.8339	0.5252	0.3466
	120	ρ	-0.0016	-0.0014	0.0010	0.0100	0.0278	0.0478
		Pval	0.9803	0.9828	0.9877	0.8751	0.6615	0.4516
	130	ρ	-0.0065	-0.0063	-0.0100	-0.0102	0.0092	0.0267
		Pval	0.9190	0.9214	0.8750	0.8726	0.8848	0.6749
	140	ρ	-0.0073	-0.0063	-0.0043	-0.0012	0.0235	0.0353
		Pval	0.9092	0.9214	0.9459	0.9852	0.7114	0.5788
DGS	50	ρ	-0.0049	0.0167	0.0427	0.1682	0.3042	0.4631
		Pval	0.9386	0.7933	0.5013	**0.0077**	**0.0000**	**0.0000**
	60	ρ	-0.0014	0.0179	0.0480	0.1771	0.4376	0.5867
		Pval	0.9828	0.7779	0.4498	**0.0050**	**0.0000**	**0.0000**
	70	ρ	-0.0069	0.0096	0.0374	0.1105	0.3708	0.5471
		Pval	0.9133	0.8797	0.5556	0.0811	**0.0000**	**0.0000**
	80	ρ	-0.0219	-0.0097	0.0143	0.1206	0.3565	0.5134
		Pval	0.7298	0.8785	0.8224	0.0570	**0.0000**	**0.0000**
	90	ρ	-0.0110	0.0035	0.0310	0.1036	0.3644	0.5214
		Pval	0.8622	0.9562	0.6259	0.1021	**0.0000**	**0.0000**
	100	ρ	-0.0082	-0.0004	0.0276	0.0896	0.2884	0.4841
		Pval	0.8967	0.9951	0.6644	0.1579	**0.0000**	**0.0000**
	110	ρ	-0.0132	-0.0029	0.0223	0.0948	0.2721	0.5075
		Pval	0.8354	0.9639	0.7261	0.1349	**0.0000**	**0.0000**
	120	ρ	-0.0222	-0.0098	0.0176	0.0914	0.2465	0.5282
		Pval	0.7268	0.8777	0.7821	0.1497	**0.0001**	**0.0000**
	130	ρ	-0.0203	-0.0052	0.0223	0.1011	0.2557	0.5076
		Pval	0.7496	0.9349	0.7252	0.1110	**0.0000**	**0.0000**
	140	ρ	-0.0121	0.0035	0.0333	0.1143	0.2646	0.5792
		Pval	0.8491	0.9560	0.5999	0.0712	**0.0000**	**0.0000**

Table A2: Correlation between start-up size and APFD based on Spearman's rank for CW case study.

IT Case Study							
Alg	TS Size	Stat	Start-up Size				
			1	5	10	20	50
DAS	50	ρ	0.0046	0.0046	0.0046	0.0252	0.0939
		Pval	0.9366	0.9366	0.9366	0.6634	0.1044
	60	ρ	0.0244	0.0244	0.0244	0.0346	0.0776
		Pval	0.6734	0.6734	0.6734	0.5508	0.1802
	70	ρ	-0.0151	-0.0151	-0.0151	-0.0020	0.0570
		Pval	0.7941	0.7941	0.7941	0.9721	0.3248
	80	ρ	0.0283	0.0283	0.0283	0.0362	0.1093
		Pval	0.6252	0.6252	0.6252	0.5319	0.0587
	90	ρ	0.0147	0.0147	0.0147	0.0255	0.0697
		Pval	0.7996	0.7996	0.7996	0.6606	0.2290
	100	ρ	0.0617	0.0617	0.0617	0.0686	0.1101
		Pval	0.2867	0.2867	0.2867	0.2359	0.0567
	110	ρ	0.0675	0.0675	0.0675	0.0698	0.0965
		Pval	0.2437	0.2437	0.2437	0.2278	0.0951
	120	ρ	0.0454	0.0454	0.0454	0.0441	0.0953
		Pval	0.4330	0.4330	0.4330	0.4465	0.0995
	130	ρ	0.0625	0.0625	0.0625	0.0675	0.0900
		Pval	0.2809	0.2809	0.2809	0.2437	0.1199
	140	ρ	0.0408	0.0408	0.0408	0.0414	0.0586
		Pval	0.4811	0.4811	0.4811	0.4750	0.3117
DGS	50	ρ	0.2454	0.2808	0.2984	0.3266	0.3605
		Pval	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
	60	ρ	0.1330	0.1710	0.1996	0.2163	0.2490
		Pval	**0.0212**	**0.0030**	**0.0005**	**0.0002**	**0.0000**
	70	ρ	0.0944	0.1436	0.1673	0.2121	0.2279
		Pval	0.1027	**0.0128**	**0.0037**	**0.0002**	**0.0001**
	80	ρ	0.0610	0.1066	0.1364	0.1696	0.2115
		Pval	0.2921	0.0653	**0.0181**	**0.0032**	**0.0002**
	90	ρ	0.0864	0.1471	0.1729	0.2144	0.2214
		Pval	0.1356	**0.0107**	**0.0027**	**0.0002**	**0.0001**
	100	ρ	0.0414	0.0894	0.1320	0.1667	0.2005
		Pval	0.4750	0.1222	**0.0222**	**0.0038**	**0.0005**
	110	ρ	0.0108	0.0608	0.0995	0.1255	0.1455
		Pval	0.8520	0.2939	0.0854	**0.0298**	**0.0117**
	120	ρ	0.0717	0.1141	0.1456	0.1659	0.1905
		Pval	0.2158	**0.0483**	**0.0116**	**0.0040**	**0.0009**
	130	ρ	0.0335	0.0811	0.1008	0.1377	0.1605
		Pval	0.5638	0.1613	0.0813	**0.0170**	**0.0053**
	140	ρ	0.1106	0.1614	0.1786	0.1914	0.1939
		Pval	0.0556	**0.0051**	**0.0019**	**0.0009**	**0.0007**

Table A3: Correlation between reallocation size and APFD based on Spearman's rank for IT case study.

CW Case Study							
Alg	TS Size	Stat	Start-up Size				
			1	5	10	20	50
DAS	50	ρ	-0.0819	-0.0819	-0.0819	-0.0819	-0.0255
		Pval	0.1572	0.1572	0.1572	0.1572	0.6604
	60	ρ	-0.1093	-0.1093	-0.1093	-0.1093	0.0379
		Pval	0.0587	0.0587	0.0587	0.0587	0.5132
	70	ρ	-0.1640	-0.1640	-0.1640	-0.1640	-0.0589
		Pval	**0.0044**	**0.0044**	**0.0044**	**0.0044**	0.3096
	80	ρ	-0.0954	-0.0954	-0.0954	-0.0954	-0.0291
		Pval	0.0992	0.0992	0.0992	0.0992	0.6162
	90	ρ	-0.0825	-0.0825	-0.0825	-0.0825	-0.0249
		Pval	0.1538	0.1538	0.1538	0.1538	0.6679
	100	ρ	-0.1168	-0.1168	-0.1168	-0.1168	-0.0384
		Pval	**0.0433**	**0.0433**	**0.0433**	**0.0433**	0.5077
	110	ρ	-0.1156	-0.1156	-0.1156	-0.1156	-0.0303
		Pval	**0.0454**	**0.0454**	**0.0454**	**0.0454**	0.6010
	120	ρ	-0.0875	-0.0875	-0.0875	-0.0875	-0.0213
		Pval	0.1305	0.1305	0.1305	0.1305	0.7129
	130	ρ	0.0102	0.0102	0.0102	0.0102	0.0382
		Pval	0.8604	0.8604	0.8604	0.8604	0.5094
	140	ρ	-0.0646	-0.0646	-0.0646	-0.0646	-0.0050
		Pval	0.2648	0.2648	0.2648	0.2648	0.9308
DGS	50	ρ	-0.4247	-0.3228	-0.2043	-0.0076	0.0037
		Pval	**0.0000**	**0.0000**	**0.0004**	0.8964	0.9489
	60	ρ	-0.5118	-0.4042	-0.2588	-0.0163	0.0364
		Pval	**0.0000**	**0.0000**	**0.0000**	0.7785	0.5299
	70	ρ	-0.4917	-0.4116	-0.2726	-0.0572	-0.0187
		Pval	**0.0000**	**0.0000**	**0.0000**	0.3231	0.7468
	80	ρ	-0.4813	-0.3634	-0.2894	-0.0566	-0.0111
		Pval	**0.0000**	**0.0000**	**0.0000**	0.3288	0.8483
	90	ρ	-0.5052	-0.3762	-0.2737	-0.0431	-0.0073
		Pval	**0.0000**	**0.0000**	**0.0000**	0.4565	0.9001
	100	ρ	-0.4304	-0.3257	-0.2158	-0.0311	0.0022
		Pval	**0.0000**	**0.0000**	**0.0002**	0.5917	0.9704
	110	ρ	-0.4314	-0.3111	-0.2059	-0.0288	0.0208
		Pval	**0.0000**	**0.0000**	**0.0003**	0.6187	0.7193
	120	ρ	-0.4349	-0.3279	-0.2279	-0.0375	0.0082
		Pval	**0.0000**	**0.0000**	**0.0001**	0.5180	0.8872
	130	ρ	-0.4121	-0.3220	-0.2269	-0.0419	0.0086
		Pval	**0.0000**	**0.0000**	**0.0001**	0.4701	0.8821
	140	ρ	-0.4408	-0.3807	-0.2705	-0.0649	-0.0114
		Pval	**0.0000**	**0.0000**	**0.0000**	0.2627	0.8434

Table A4: Correlation between reallocation size and APFD based on Spearman's rank for CW case study.

Bibliography

[AB11] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 1–10, New York, NY, USA, 2011. ACM.

[ABC+13] Saswat Anand, Edmund K Burke, Tsong Yueh Chen, John Clark, Myra B Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil McMinn, Antonia Bertolino, et al. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, 2013.

[ABHPW10] Shaukat Ali, Lionel C. Briand, Hadi Hemmati, and Rajwinder Kaur Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Trans. Softw. Eng.*, 36(6):742–762, November 2010.

[ABKS16] Sven Apel, Don Batory, Christian Kstner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[AF11] Andrea Arcuri and Gordon Fraser. *On Parameter Tuning in Search Based Software Engineering*, pages 33–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[AHLL+17] M. Al-Hajjaji, S. Lity, R. Lachmann, T. Thüm, I. Schaefer, and G. Saake. Delta-oriented product prioritization for

similarity-based product-line testing. In *VACE '17*, pages 34–40, 2017.

[AHTL+16] Mustafa Al-Hajjaji, Thomas Thüm, Malte Lochau, Jens Meinicke, and Gunter Saake. Effective product-line testing using similarity-based product prioritization. *Software & Systems Modeling*, pages 1–23, 2016.

[AHTM+14] Mustafa Al-Hajjaji, Thomas Thüm, Jens Meinicke, Malte Lochau, and Gunter Saake. Similarity-based prioritization in software product-line testing. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 197–206, New York, NY, USA, 2014. ACM.

[AIH15] Sara Abbaspour Asadollah, Rafia Inam, and Hans Hansson. A survey on testing for cyber physical system. In *IFIP International Conference on Testing Software and Systems*, pages 194–207. Springer, 2015.

[AKM20] Wesley KG Assunção, Jacob Krüger, and Willian DF Mendonça. Variability management meets microservices: six challenges of re-engineering microservice-based webshops. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*, pages 1–6, 2020.

[ALHR17] Mathieu Acher, Roberto E. Lopez-Herrejon, and Rick Rabiser. Teaching software product lines: A snapshot of current practices and challenges. *ACM Trans. Comput. Educ.*, 18(1):2:1–2:31, October 2017.

[AMAAA17] Mohammed Nasser Al-Mhiqani, Rabiah Ahmad, Karrar Hameed Abdulkareem, and Nabeel Salih Ali. Investigation study of cyber-physical systems: Characteristics, application domains, and security challenges. *ARPN Journal of Engineering and Applied Sciences*, 12(22):6557–6567, 2017.

[AME17] Aitor Arrieta, Urtzi Markiegi, and Leire Etxeberria. Towards mutation testing of configurable simulink models:

a product line engineering perspective. Technical report, Mondragon Unibertsitatea, 2017.

[ANBS18] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1016–1026. IEEE, 2018.

[AO08] P. Ammann and J. Offutt. *Introduction to Software Testing*. First edition edition, 2008.

[AO16] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, second edition edition, December 2016.

[AOCN20] Ana Paula Allian, Edson OliveiraJr, Rafael Capilla, and Elisa Yumi Nakagawa. Have variability tools fulfilled the needs of the software industry? *Journal of Universal Computer Science*, 26(10):1282–1311, 2020.

[Arr17] Aitor Arrieta. *Simulation-based testing of highly configurable cyber-physical systems: automation, optimization and debugging*. phdthesis, Mondragon Unibertsitatea, December 2017.

[ASE15] Aitor Arrieta, Goiuria Sagardui, and Leire Etxeberria. Test control algorithms for the validation of cyber-physical systems product lines. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 273–282, New York, NY, USA, 2015. ACM.

[ASEZ16] Aitor Arrieta, Goiuria Sagardui, Leire Etxeberria, and Justyna Zander. Automatic generation of test system instances for configurable cyber-physical systems. *Software Quality Journal*, pages 1–43, 2016.

[AWA+18] Aitor Arrieta, Shuai Wang, Ainhoa Arruabarrena, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. Multi-objective black-box test case selection for cost-effectively testing simulation models. In *Proceedings of the Genetic*

and Evolutionary Computation Conference, GECCO '18, pages 1411–1418, New York, NY, USA, 2018. ACM.

[AWM+17a] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. Employing multi-objective search to enhance reactive test case generation and prioritization for testing industrial cyber-physical systems. *IEEE Transactions on Industrial Informatics*, 14(3):1055–1066, 2017.

[AWM+17b] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. Search-based test case generation for cyber-physical systems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, 2017.

[AWM+19] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Ainhoa Arruabarrena, Leire Etxeberria, and Goiuria Sagardui. Pareto efficient multi-objective black-box test case selection for simulation-based testing. *Information & Software Technology*, 114:137–154, 2019.

[AWSE16a] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. Search-based test case selection of cyber-physical system product lines for simulation-based validation. In *Proceedings of the 20th International Systems and Software Product Line Conference*, SPLC '16, pages 297–306, New York, NY, USA, 2016. ACM.

[AWSE16b] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. Test case prioritization of configurable cyber-physical systems with weight-based search algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 1053–1060, New York, NY, USA, 2016. ACM.

[AWSE19] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. Search-based test case prioritization for simulation-based testing of cyber-physical system product lines. *Journal of Systems and Software*, 149:1 – 34, 2019.

[Bat05] Don Batory. Feature models, grammars, and propositional formulas. In Henk Obbink and Klaus Pohl, editors, *Software Product Lines: 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005. Proceedings*, pages 7–20, Berlin, Heidelberg, 2005. Springer, Springer Berlin Heidelberg.

[BBG+88] DS Batory, JR Barnett, Jorge F Garza, Kenneth Paul Smith, K Tsukuda, Brian C Twichell, and TE Wise. Genesis: An extensible database management system. *IEEE Transactions on Software Engineering*, 14(11):1711–1730, 1988.

[BBSR16] Ebrahim Bagheri, David Benavides, Klaus Schmid, and Per Runeson. Foreword to the special issue on empirical evidence on software product line engineering. *Empirical Software Engineering*, 21(4):1579–1585, 2016.

[BC07] Renée C. Bryce and Charles J. Colbourn. One-test-at-a-time heuristic search for interaction test suites. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 1082–1089, New York, NY, USA, 2007. ACM.

[BLLS14] H. Baller, S. Lity, M. Lochau, and I. Schaefer. Multi-objective test suite optimization for incremental product family testing. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, pages 303–312, March 2014.

[BNSB16] Lionel Briand, Shiva Nejati, Mehrdad Sabetzadeh, and Domenico Bianculli. Testing the untestable: Model testing of complex software-intensive systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 789–792, New York, NY, USA, 2016. ACM.

[Boe79] BW Boehm. Guidelines for verifying and validating software requirements and design specifications, pa samet. In *Proc. of the EURO IFIP*, volume 79, 1979.

[Bos01] Jan Bosch. Software product lines: Organizational alternatives. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 91–100, Washington, DC, USA, 2001. IEEE Computer Society.

[BSC10] David Benavides, Sergio Segura, and Antonio Ruiz Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, 2010.

[BSL+10] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. Variability modeling in the real: A perspective from the operating systems domain. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 73–82, New York, NY, USA, 2010. ACM.

[BSL+13] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, and Krzysztof Czarnecki. A study of variability models and languages in the systems software domain. *IEEE Transactions on Software Engineering*, 39(12):1611–1640, 2013.

[BSR03] Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer. Scaling step-wise refinement. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 187–197, Washington, DC, USA, 2003. IEEE Computer Society.

[BSZ+20] Thorsten Berger, Jan-Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, and Jabier Martinez. The state of adoption and the challenges of systematic variability management in industry. *Empirical Software Engineering*, 25(3):1755–1797, 2020.

[BX16] Benjamin Busjaeger and Tao Xie. Learning for test prioritization: an industrial case study. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 975–980. ACM, 2016.

[CDFP97] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The aetg system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, Jul 1997.

[CDS07] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Interaction testing of highly-configurable systems in the presence of constraints. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ISSTA '07, pages 129–139, New York, NY, USA, 2007. ACM.

[CDS08] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Trans. Softw. Eng.*, 34(5):633–650, September 2008.

[CGHX19] Yuntianyi Chen, Yongfeng Gu, Lulu He, and Jifeng Xuan. Regression models for performance ranking of configurable systems: A comparative study. In *International Workshop on Structured Object-Oriented Formal Language and Method*, pages 243–258. Springer, 2019.

[Che17a] Hong Chen. Applications of cyber-physical system: a literature review. *Journal of Industrial Integration and Management*, 2(03):1750012, 2017.

[Che17b] Hong Chen. Theoretical foundations for cyber-physical systems: a literature review. *Journal of Industrial Integration and Management*, 2(03):1750013, 2017.

[CKMT10] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*, 83:60 – 66, 2010.

[CLGGB20] Ana Eva Chacón-Luna, Antonio Manuel Gutiérrez, José A Galindo, and David Benavides. Empirical software product line engineering: a systematic literature review. *Information and Software Technology*, 128:106389, 2020.

[CM13] Cagatay Catal and Deepti Mishra. Test case prioritization: a systematic mapping study. *Software Quality Journal*, 21(3):445–478, Sep 2013.

[CMM+18] Shafiul Azam Chowdhury, Soumik Mohian, Sidharth Mehra, Siddhant Gawsane, Taylor T. Johnson, and Christoph Csallner. Automatically finding bugs in a commercial cyber-physical system development tool chain with slforge. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 981–992, New York, NY, USA, 2018. ACM.

[CN01] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, August 2001.

[dCMMCDA14] Ivan do Carmo Machado, John D McGregor, Yguaratã Cerqueira Cavalcanti, and Eduardo Santana De Almeida. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology*, 56(10):1183–1199, 2014.

[DH18] Ashraf Darwish and Aboul Ella Hassanien. Cyber physical systems design, methodology, and integration: the current status and future outlook. *Journal of Ambient Intelligence and Humanized Computing*, 9(5):1541–1556, 2018.

[DLV12] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli. Modeling cyber-physical systems. *Proceedings of the IEEE*, 100(1):13–28, Jan 2012.

[dMSNdCMM+11] Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, John D. McGregor, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. A systematic mapping study of software product lines testing. *Information and Software Technology*, 53(5):407–423, may 2011.

[DNABL13] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche. Coverage-based test case prioritisation: An industrial case study. In *Sixth International Conference on Software Testing, Verification and Validation*, pages 302 – 11, Los Alamitos, CA, USA, 2013.

[DPC+14] Xavier Devroey, Gilles Perrouin, Maxime Cordy, Pierre-Yves Schobbens, Axel Legay, and Patrick Heymans. Towards statistical prioritization for software product lines testing. In *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, page 10. ACM, 2014.

[DPC+15] Xavier Devroey, Gilles Perrouin, Maxime Cordy, Hamza Samih, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. Statistical prioritization for software product line testing: an experience report. *Software & Systems Modeling*, pages 1–19, 2015.

[DRB+13] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. An exploratory study of cloning in industrial software product lines. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 25–34. IEEE, 2013.

[EBA+11] Alireza Ensan, Ebrahim Bagheri, Mohsen Asadi, Dragan Gasevic, and Yevgen Biletskiy. Goal-oriented test case selection and prioritization for product line feature models. In *Proceedings of the 2011 Eighth International Conference on Information Technology: New Generations*, ITNG '11, pages 291–298, Washington, DC, USA, 2011. IEEE Computer Society.

[EBG12] Faezeh Ensan, Ebrahim Bagheri, and Dragan Gašević. Evolutionary search-based test generation for software product line feature models. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering*, CAiSE'12, pages 613–628, Berlin, Heidelberg, 2012. Springer-Verlag.

[ELM+17] Leire Etxeberria, Felix Larrinaga, Urtzi Markiegi, Aitor Arrieta, and Goiuria Sagardui. Enabling co-simulation of smart energy control systems for buildings and districts. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 2017.

[EMR02] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: a family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, Feb 2002.

[ER11] Emelie Engström and Per Runeson. Software product line testing–a systematic mapping study. *Information and Software Technology*, 53(1):2–13, 2011.

[ERL11] Emelie Engström, Per Runeson, and Andreas Ljung. Improving regression testing transparency and efficiency with history-based prioritization–an industrial case study. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 367–376. IEEE, 2011.

[ERP14] Sebastian Elbaum, Gregg Rothermel, and John Penix. Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*, pages 235–245. ACM, 2014.

[ESYES19] Sascha El-Sharkawy, Nozomi Yamagishi-Eichler, and Klaus Schmid. Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. *Information and Software Technology*, 106:1–30, 2019.

[EYHB15] Michael G. Epitropakis, Shin Yoo, Mark Harman, and Edmund K. Burke. Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 234–245, New York, NY, USA, 2015. ACM.

[FCWZ14] Chunrong Fang, Zhenyu Chen, Kun Wu, and Zhihong Zhao. Similarity-based test case prioritization using ordered sequences of program entities. *Software Quality Journal*, 22(2):335–361, 2014.

[FLHE18] Stefan Fischer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. Towards a fault-detection benchmark for evaluating software product line testing approaches. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 2034–2041, 2018.

[FMR+20] Stefan Fischer, Gabriela Karoline Michelon, Rudolf Ramler, Lukas Linsbauer, and Alexander Egyed. Automated test reuse for highly configurable software. *Empirical Software Engineering*, 25(6):5295–5332, 2020.

[FPCY16] R. Feldt, S. Poulding, D. Clark, and S. Yoo. Test set diameter: Quantifying the diversity of sets of test cases. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 223–233, April 2016.

[FR19] Gordon Fraser and José Miguel Rojas. Software testing. In *Handbook of Software Engineering*, pages 123–192. Springer, 2019.

[GBT21] Didem Gürdür Broo, Ulf Boman, and Martin Törngren. Cyber-physical systems research and education in 2030: Scenarios and strategies. *Journal of Industrial Information Integration*, 21:100192, 2021.

[GCD11] Brady J. Garvin, Myra B. Cohen, and Matthew B. Dwyer. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering*, 16(1):61–102, 2011.

[GSB+19] Sergio García, Daniel Strüber, Davide Brugali, Alessandro Di Fava, Philipp Schillinger, Patrizio Pelliccione, and Thorsten Berger. Variability modeling of service robots: Experiences and challenges. In *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems*, VAMOS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[HAB13] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. Achieving scalable model-based testing through test case diversity.

ACM Trans. Softw. Eng. Methodol., 22(1):6:1–6:42, March 2013.

[Ham50] Rw Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, 1950.

[HBT16] Le Thi My Hanh, Nguyen Thanh Binh, and Khuat Thanh Tung. A novel fitness function of metaheuristic algorithms for test data generation for simulink models based on mutation analysis. *Journal of Systems and Software*, 120:17 – 30, 2016.

[HFM15] H. Hemmati, Z. Fang, and M. V. Mäntylä. Prioritizing manual test cases in traditional and rapid release environments. In *Proceedings of the 8th International Conference on Software Testing, Verification and Validation (ICST'15)*, pages 1–10, 2015.

[HGPB20] Ines Hajri, Arda Goknil, Fabrizio Pastore, and Lionel C Briand. Automating system test case classification and prioritization for use case-driven testing in product lines. *Empirical Software Engineering*, 25(5):3711–3769, 2020.

[HJK+14] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang. Search based software engineering for software product line engineering: A survey and directions for future work. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 5–18, New York, NY, USA, 2014. ACM.

[HLL+16] Robert M. Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. Sip: Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Trans. Softw. Eng. Methodol.*, 25(2):17:1–17:39, April 2016.

[HMZ12] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, December 2012.

[HPF19] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. Software product line engineering: a practical experience. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*, pages 164–176, 2019.

[HPH12] Yu-Chi Huang, Kuan-Li Peng, and Chin-Yu Huang. A history-based cost-cognizant test case prioritization technique in regression testing. *Journal of Systems and Software*, 85(3):626–637, 2012.

[HPH+16] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. Comparing white-box and black-box test prioritization. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 523–534, New York, NY, USA, 2016. ACM.

[HPHT15] Christopher Henard, Mike Papadakis, Mark Harmany, and Yves Le Traon. Combining multi-objective search and constraint solving for configuring large scale software product lines. In *37th International Conference on Software Engineering (ICSE'15)*, pages 517–528, 2015.

[HPLT14] Christopher Henard, Mike Papadakis, and Yves Le Traon. *Mutation-Based Generation of Software Product Line Test Configurations*, pages 92–106. Springer International Publishing, Cham, 2014.

[HPMFA+16] Ruben Heradio, Hector Perez-Morago, David Fernandez-Amoros, Francisco Javier Cabrerizo, and Enrique Herrera-Viedma. A bibliometric analysis of 20 years of research on software product lines. *Information and Software Technology*, 72:1–15, 2016.

[HPP+13a] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. Assessing software product line testing via model-based mutation: An application to similarity testing. In *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, pages 188–197. IEEE, 2013.

[HPP+13b] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. Multi-objective test generation for software product lines. In *Proceedings of the 17th International Software Product Line Conference*, pages 62–71. ACM, 2013.

[HPP+14] C Henard, M Papadakis, G Perrouin, J Klein, P Heymans, and Y Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Trans. Software Eng.*, 40(7):650–670, 2014.

[HVCR01] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson. A practical tutorial on modified condition/decision coverage. Technical report, NASA, 2001.

[HWJ13] Kagermann H., Wahlster W., and Helbig J. Recommendations for implementing the strategic initiative industrie 4.0. Technical report, acatech, National Academy of Science and Engineering, 2013.

[HZS+16] Claus Hunsen, Bo Zhang, Janet Siegmund, Christian Kästner, Olaf Leßenich, Martin Becker, and Sven Apel. Preprocessor-based variability in open-source and industrial software systems: An empirical study. *Empirical Software Engineering*, 21(2):449–482, 2016.

[HZZ+16] D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu, and T. Xie. To be optimal or not in test-case prioritization. *IEEE Transactions on Software Engineering*, 42(5):490–505, May 2016.

[Jaz14] N. Jazdi. Cyber physical systems in the context of industry 4.0. In *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 1–4, May 2014.

[JC15] Bo Jiang and W.K. Chan. Input-based adaptive randomized test case prioritization: A local beam search approach. *Journal of Systems and Software*, 105(0):91 – 106, 2015.

[JCL11] Jeff C Jensen, Danica H Chang, and Edward A Lee. A model-based design methodology for cyber-physical systems. In *2011 7th international wireless communications and mobile computing conference*, pages 1666–1671. IEEE, 2011.

[JH03] James A Jones and Mary Jean Harrold. Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE Transactions on software Engineering*, 29(3):195–209, 2003.

[JH10] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering*, 37(5):649–678, 2010.

[JHF11a] Martin Fagereng Johansen, Oystein Haugen, and Franck Fleurey. Properties of realistic feature models make combinatorial testing of product lines feasible. In *Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems*, MODELS'11, pages 638–652, Berlin, Heidelberg, 2011. Springer-Verlag.

[JHF11b] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. A survey of empirics of strategies for software product line testing. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 266–269. IEEE, 2011.

[JHF12] Martin Fagereng Johansen, Oystein Haugen, and Franck Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC '12, pages 46–55, New York, NY, USA, 2012. ACM.

[JHQJ08] Li Jin-Hua, Li Qiong, and Li Jing. The w-model for testing software product lines. In *2008 International Symposium on Computer Science and Computational Technology*, volume 1, pages 690–693. IEEE, 2008.

[JJI+14] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 654–665, New York, NY, USA, 2014. ACM.

[KAF12] Alireza Khalilian, Mohammad Abdollahi Azgomi, and Yalda Fazlalizadeh. An improved method for test case prioritization by incorporating historical test case data. *Science of Computer Programming*, 78(1):93–116, 2012.

[KAuR+09] Christian Kästner, Sven Apel, Syed Saif ur Rahman, Marko Rosenmüller, Don Batory, and Gunter Saake. On the impact of the optional feature problem: Analysis and case studies. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 181–190, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[KBK11] C. H. P. Kim, D. S. Batory, and S. Khurshid. Reducing combinatorics in testing product lines. In *AOSD '11*, pages 57–68, 2011.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-Oriented domain analysis (FODA) feasibility study. Technical report, 1990.

[KDJ+16] James Kapinski, Jyotirmoy V Deshmukh, Xiaoqing Jin, Hisahiro Ito, and Ken Butts. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems Magazine*, 36(6):45–64, 2016.

[KIJT18] Muhammad Khatibsyarbini, Mohd Adham Isa, Dayang N. A. Jawawi, and Rooster Tumeng. Test case prioritization approaches in regression testing: A systematic literature review. *Information and Software Technology*, 93:74 – 93, 2018.

[Kim17] Jin Ho Kim. A review of cyber-physical system research relevant to the emerging it trends: industry 4.0, iot, big data, and cloud computing. *Journal of industrial integration and management*, 2(03):1750011, 2017.

[KKLH09] Rick Kuhn, Raghu Kacker, Yu Lei, and Justin Hunter. Combinatorial software testing. *Computer*, 42:94–96, 2009.

[KKT08] Bogdan Korel, George Koutsogiannakis, and Luay H Tahat. Application of system models in regression test suite prioritization. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 247–256. IEEE, 2008.

[KKW+18] Sebastian Krieter, Jacob Krüger, Nico Weichbrodt, Vasily A Sartakov, Rüdiger Kapitza, and Thomas Leich. Towards secure dynamic product lines in the cloud. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 5–8, 2018.

[KM03] R Kolb and D Muthig. Challenges in testing software product lines. In *Proceedings of CONQUEST*, volume 3, pages 81–95, 2003.

[KMTG18] J. Krüger, Al-Hajjaji M., Leich T., and Saake G. Mutation operators for feature-oriented software product lines. *Software Testing, Verification and Reliability*, 2018.

[KNK+17] Jacob Krüger, Sebastian Nielebock, Sebastian Krieter, Christian Diedrich, Thomas Leich, Gunter Saake, Sebastian Zug, and Frank Ortmeier. Beyond software product lines: Variability modeling in cyber-physical systems. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*, pages 237–241, 2017.

[KP02] Jung-Min Kim and Adam Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *Proceedings of the 24th international conference on software engineering*, pages 119–129. ACM, 2002.

[KSK+19] Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, Alexander Grebhahn, and Sven Apel. Tradeoffs in modeling performance of highly configurable software systems. *Software & Systems Modeling*, 18(3):2265–2283, 2019.

[KTH05] Bogdan Korel, Luay Ho Tahat, and Mark Harman. Test prioritization using system models. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 559–568. IEEE, 2005.

[LAHTS17] Sascha Lity, Mustafa Al-Hajjaji, Thomas Thüm, and Ina Schaefer. Optimizing product orders using graph algorithms for improving incremental product-line analysis. In *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems*, VAMOS '17, pages 60–67, New York, NY, USA, 2017. ACM.

[LBB15] Zhou Lu, MM Bezemer, and JF Broenink. Model-driven design of simulation support for the terra robot software tool suite. *Communicating process architectures 2015*, pages 143–158, 2015.

[LBK15] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18 – 23, 2015.

[LBL+17] Remo Lachmann, Simon Beddig, Sascha Lity, Sandro Schulze, and Ina Schaefer. Risk-based integration testing of software product lines. In *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems*, VAMOS '17, pages 52–59, New York, NY, USA, 2017. ACM.

[LCTK13] Chu-Ti Lin, Cheng-Ding Chen, Chang-Shi Tsai, and Gregory M Kapfhammer. History-based test case prioritization with software version awareness. In *2013 18th International Conference on Engineering of Complex Computer Systems*, pages 171–172. IEEE, 2013.

[LFN+17] R. Lachmann, M. Felderer, M. Nieke, S. Schulze, C. Seidl, and I. Schaefer. Multi-objective black-box test case selec-

tion for system testing. In *GECCO '17*, pages 1311–1318. ACM, 2017.

[LHFC+14] Roberto E Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Lukas Linsbauer, Alexander Egyed, and Enrique Alba. A hitchhiker's guide to search-based software engineering for software product lines. *Computer Research Repositori (CoRR)*, 2014.

[LHFC+16] Roberto E Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Alexander Egyed, and Enrique Alba. Evolutionary computation for software product line testing: an overview and open challenges. In *Computational Intelligence and Quantitative Software Engineering*, pages 59–87. Springer, 2016.

[LHFRE15] Roberto E Lopez-Herrejon, Stefan Fischer, Rudolf Ramler, and Alexander Egyed. A first systematic mapping study on combinatorial interaction testing for software product lines. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, pages 1–10. IEEE, 2015.

[LHLE15] Roberto E. Lopez-Herrejon, Lukas Linsbauer, and Alexander Egyed. A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology*, 61:33 – 51, 2015.

[LKJ20] Jihyun Lee, Sungwon Kang, and Pilsu Jung. Test coverage criteria for software product line testing: Systematic literature review. *Information and Software Technology*, page 106272, 2020.

[LKL12] Jihyun Lee, Sungwon Kang, and Danhyung Lee. A survey on software product line testing. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 31–40. ACM, 2012.

[LLL+15] Remo Lachmann, Sascha Lity, Sabrina Lischke, Simon Beddig, Sandro Schulze, and Ina Schaefer. Delta-oriented test case prioritization for integration testing of software

product lines. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 81–90, New York, NY, USA, 2015. ACM.

[LLN+16] Bing Liu, Lucia, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Simulink fault localization: an iterative statistical debugging approach. *Software Testing, Verification and Reliability*, 26(6):431–459, 2016.

[LLNB17] Bing Liu, Lucia Lucia, Shiva Nejati, and Lionel Briand. Improving fault localization for simulink models using search-based testing and prediction models. In *24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2017)*, 2017.

[LMP16] Qi Luo, Kevin Moran, and Denys Poshyvanyk. A large-scale empirical comparison of static and dynamic test case prioritization techniques. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 559–570. ACM, 2016.

[LMZP19] Q. Luo, K. Moran, L. Zhang, and D. Poshyvanyk. How do static and dynamic test case prioritization techniques perform on modern software systems? an extensive study on github projects. *IEEE Transactions on Software Engineering*, 45(11):1054–1080, Nov 2019.

[LNLB19] Bing Liu, Shiva Nejati, Lucia, and Lionel C. Briand. Effective fault localization of automotive simulink models: achieving the trade-off between test oracle effort and fault localization accuracy. *Empirical Software Engineering*, 24(1):444–490, Feb 2019.

[LPBM12] Yves Ledru, Alexandre Petrenko, Sergiy Boroday, and Nadine Mandran. Prioritizing test cases with string distances. *Automated Software Engineering*, 19(1):65–95, 2012.

[LS14] Hartmut Lackner and Martin Schmidt. Towards the assessment of software product line tests: A mutation system for variable systems. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for*

Workshops, Demonstrations and Tools - Volume 2, SPLC '14, pages 62–69, New York, NY, USA, 2014. ACM.

[LS17] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. MIT Press Ltd, second edition edition, 2017.

[LSR07] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[LTMHT14] Khuat Thanh Le Thi My Hanh and Nguyen Thanh Binh Tung. Mutation-based test data generation for simulink models using genetic algorithm and simulated annealing. *International Journal of Computer and Information Technology*, 3(04):763–771, 2014.

[LUV09] Beatriz P'erez Lamancha, Macario Polo Usaola, and Mario Piattini Velthius. Software product line testing: A systematic review. *ICSOFT (1)*, pages 23–30, 2009.

[LYA20] Hong Lu, Tao Yue, and Shaukat Ali. Pattern-based interactive configuration derivation for cyber-physical system product lines. *ACM Transactions on Cyber-Physical Systems*, 4(4):1–24, 2020.

[MAES19a] Urtzi Markiegi, Aitor Arrieta, Leire Etxeberria, and Goiuria Sagardui. Test case selection using structural coverage in software product lines for time-budget constrained scenarios. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 2362–2371, 2019.

[MAES19b] Urtzi Markiegi, Aitor Arrieta, Leire Etxeberria, and Goiuria Sagardui. White-box and black-box test quality metrics for configurable simulation models. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume B, Paris, France, September 9-13, 2019*, pages 92:1–92:4. ACM, 2019.

[MASE17] Urtzi Markiegi, Aitor Arrieta, Goiuria Sagardui, and Leire Etxeberria. Search-based product line fault detection allocating test cases iteratively. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume A*, SPLC '17, pages 123–132, New York, NY, USA, 2017. ACM.

[Mat21] The mathworks, inc. - simulink tool. https://www.mathworks.com/products/simulink.html, 2021. Last accessed: 2021-04-15.

[MB03] M. Marré and A. Bertolino. Using spanning sets for coverage testing. *IEEE Transactions on Software Engineering*, 29:974–984, 2003.

[McG01] John D McGregor. Testing a software product line. techreport CMU/SEI-2001-TR-022, Software Engineering Institute, Carnegie Mellon University, December 2001.

[MCVB18] Breno Miranda, Emilio Cruciani, Roberto Verdecchia, and Antonia Bertolino. Fast approaches to scalable similarity-based test case prioritization. In *Proceedings of the 40th International Conference on Software Engineering*, pages 222–232. ACM, 2018.

[MGS13] Dusica Marijan, Arnaud Gotlieb, and Sagar Sen. Test case prioritization for continuous regression testing: An industrial case study. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance (ICSM'13)*, pages 540–543. IEEE Computer Society, 2013.

[MGSH13] Dusica Marijan, Arnaud Gotlieb, Sagar Sen, and Aymeric Hervieu. Practical pairwise testing for software product lines. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, pages 227–235, New York, NY, USA, 2013. ACM.

[MM63] J. C. Miller and C. J. Maloney. Systematic mistake analysis of digital computer programs. *Communications of the ACM*, 6:58–63, 1963.

[MNB17] Reza Matinnejad, Shiva Nejati, and Lionel C Briand. Automated testing of hybrid simulink/stateflow controllers: industrial case studies. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 938–943, 2017.

[MNBB15] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Effective test suites for mixed discrete-continuous stateflow controllers. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 84–95, New York, NY, USA, 2015. ACM.

[MNBB16] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Automated test suite generation for time-continuous simulink models. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 595–606, New York, NY, USA, 2016. ACM.

[MNBB19] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann. Test generation and test prioritization for simulink models with dynamic behavior. *IEEE Transactions on Software Engineering*, 45(9):919–944, Sep. 2019.

[MNBP20] Claudio Menghi, Shiva Nejati, Lionel Briand, and Yago Isasi Parache. Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 372–384. IEEE, 2020.

[MP14] Andreas Metzger and Klaus Pohl. Software product line engineering and variability management: achievements and challenges. In *Proceedings of the on Future of Software Engineering*, pages 70–84. ACM, 2014.

[MSB+14] Pieter J. Mosterman, David Escobar Sanabria, Enes Bilgin, Kun Zhang, and Justyna Zander. Automating humanitarian missions with a heterogeneous fleet of vehicles. *Annual Reviews in Control*, 38(2):259–270, 2014.

[MTS+17] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. *Mastering software variability with FeatureIDE*. Springer, 2017.

[MWVK20] Jens Meinicke, Chu-Pan Wong, Bogdan Vasilescu, and Christian Kästner. Exploring differences and commonalities between feature flags and configuration options. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, pages 233–242, 2020.

[MZ16a] Pieter J Mosterman and Justyna Zander. Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems. *Software & Systems Modeling*, 15(1):5–16, 2016.

[MZ16b] Pieter J Mosterman and Justyna Zander. Industry 4.0 as a cyber-physical system study. *Software & Systems Modeling*, 15(1):17–29, 2016.

[NAY17] Phu H Nguyen, Shaukat Ali, and Tao Yue. Model-based security engineering for cyber-physical systems: A systematic mapping study. *Information and Software Technology*, 83:116–135, 2017.

[NdCMM+11] Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, John D McGregor, Eduardo Santana De Almeida, and Silvio Romero de Lemos Meira. A systematic mapping study of software product lines testing. *Information and Software Technology*, 53(5):407–423, 2011.

[NH15] Tanzeem Bin Noor and Hadi Hemmati. A similarity-based approach for test case prioritization using historical failure data. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 58–68. IEEE, 2015.

[NL11] Changhai Nie and Hareton Leung. A survey of combinatorial testing. *ACM Comput. Surv.*, 43(2):11:1–11:29, February 2011.

[NMP20] Alessia Napoleone, Marco Macchi, and Alessandro Pozzetti. A review on the characteristics of cyber-physical systems for the future smart factories. *Journal of Manufacturing Systems*, 54:305–335, 2020.

[NOM06] Robert Nilsson, Jeff Offutt, and Jonas Mellin. Test case generation for mutation-based testing of timeliness. *Electronic Notes in Theoretical Computer Science*, 164(4):97–114, 2006.

[NWF+15] Tadahiro Noguchi, Hironori Washizaki, Yoshiaki Fukazawa, Atsutoshi Sato, and Kenichiro Ota. History-based test case prioritization for black box testing using ant colony optimization. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–2. IEEE, 2015.

[OMR10] Sebastian Oster, Florian Markert, and Philipp Ritter. Automated incremental pairwise testing of software product lines. In *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*, SPLC'10, pages 196–210, Berlin, Heidelberg, 2010. Springer-Verlag.

[OWES11] Sebastian Oster, Andreas Wübbeke, Gregor Engels, and Andy Schürr. A survey of model-based software product lines testing. In *Model-based testing for embedded systems*, chapter 13, pages 339–381. CRC Press, 2011.

[OZML11] Sebastian Oster, Ivan Zorcic, Florian Markert, and Malte Lochau. Moso-polite - tool support for pairwise and model-based software product line testing. In *VaMoS*, pages 79–82, 2011.

[PBL05] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[PHH+16] Mike Papadakis, Christopher Henard, Mark Harman, Yue Jia, and Yves Le Traon. Threats to the validity of mutation-based test assessment. In *Proceedings of the 25th Interna-*

tional Symposium on Software Testing and Analysis, ISSTA 2016, pages 354–365, New York, NY, USA, 2016. ACM.

[PII+19] Hervé Panetto, Benoit Iung, Dmitry Ivanov, Georg Weichhart, and Xiaofan Wang. Challenges for the cyber-physical manufacturing enterprises of the future. *Annual Reviews in Control*, 47:200–213, 2019.

[PJHLT15] Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 936–946, Piscataway, NJ, USA, 2015. IEEE Press.

[PKZ+19] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. Mutation testing advances: an analysis and survey. In *Advances in Computers*, volume 112, pages 275–378. Elsevier, 2019.

[PMB+12] Andreas Polzer, Daniel Merschen, Goetz Botterweck, Andreas Pleuss, Jacques Thomas, Bernd Hedenetz, and Stefan Kowalewski. Managing complexity and variability of a model-based embedded software product line. *Innovations in Systems and Software Engineering*, 8(1):35–49, Mar 2012.

[POS+12] Gilles Perrouin, Sebastian Oster, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. Pairwise testing for software product lines: Comparison of two approaches. *Software Quality Journal*, 20(3-4):605–643, 2012.

[PQM+18] Leonardo Passos, Rodrigo Queiroz, Mukelabai Mukelabai, Thorsten Berger, Sven Apel, Krzysztof Czarnecki, and Jesus Padilla. A study of feature scattering in the linux kernel. *IEEE Transactions on Software Engineering*, 2018.

[ps06] pure systems. Variant management with pure::variants. techreport, pure-systems GmbH, 2006.

[PSK+10] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. Automated and scalable t-wise test case generation strategies for software product lines. In *2010 Third international conference on software testing, verification and validation*, pages 459–468. IEEE, 2010.

[PSS+16] José A. Parejo, Ana B Sánchez, Sergio Segura, Antonio Ruiz-Cortés, Roberto E. Lopez-Herrejon, and Alexander Egyed. Multi-objective test case prioritization in highly configurable systems: A case study. *Journal of Systems and Software*, pages –, 2016.

[PUS17] Thomas W. Pieber, Thomas Ulz, and Christian Steger. SystemC test case generation with the gazebo simulator. In *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. SCITEPRESS - Science and Technology Publications, 2017.

[PWA+18] Dipesh Pradhan, Shuai Wang, Shaukat Ali, Tao Yue, and Marius Liaaen. Remap: Using rule mining and multi-objective search for dynamic test case prioritization. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 46–57. IEEE, 2018.

[PWA+19] Dipesh Pradhan, Shuai Wang, Shaukat Ali, Tao Yue, and Marius Liaaen. Employing rule mining and multi-objective search for dynamic test case prioritization. *Journal of Systems and Software*, 153:86–104, 2019.

[RALS09] Marko Rosenmüller, Sven Apel, Thomas Leich, and Gunter Saake. Tailor-made data management for embedded systems: A case study on berkeley db. *Data & Knowledge Engineering*, 68(12):1493–1512, 2009.

[RBT13] Erik Rogstad, Lionel Briand, and Richard Torkar. Test case selection for black-box regression testing of database applications. *Information and Software Technology*, 55(10):1781–1795, 2013.

[RDNK16] Raj Rajkumar, Dionisio De Niz, and Mark Klein. *Cyber-physical systems*. Addison-Wesley Professional, 2016.

[RE12] Per Runeson and Emelie Engström. Chapter 7 - regression testing in software product line engineering. volume 86 of *Advances in Computers*, pages 223–263. Elsevier, 2012.

[RLSS10] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: the next computing revolution. In *Design automation conference*, pages 731–736. IEEE, 2010.

[RRKP06] Andreas Reuys, Sacha Reis, Erik Kamsties, and Klaus Pohl. The scented method for testing software product lines. In *Software Product Lines*, chapter 13, pages 479–520. Springer-Verlag Berlin Heidelberg, 2006.

[RSB+18] Rick Rabiser, Klaus Schmid, Martin Becker, Goetz Botterweck, Matthias Galster, Iris Groher, and Danny Weyns. A study and comparison of industrial vs. academic software product line research published at splc. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*, pages 14–24, 2018.

[RUCH99] Gregg Rothermel, Roland H Untch, Chengyun Chu, and Mary Jean Harrold. Test case prioritization: An empirical study. In *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*, pages 179–188. IEEE, 1999.

[RUCH01] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, 27(10):929–948, 2001.

[RWSH08] A. Rajan, M. Whalen, M. Staats, and M. Heimdahl. Requirements coverage as an adequacy measure for conformance testing. *Formal Methods and Software Engineering*, pages 86–104, 2008.

[RZ21] Rick Rabiser and Alois Zoitl. Towards mastering variability in software-intensive cyber-physical production systems. *Procedia Computer Science*, 180:50–59, 2021.

[SAM+17] Goiuria Sagardui, Joseba Agirre, Urtzi Markiegi, Aitor Arrieta, Carlos Fernando Nicolás, and Jose María Martín. Multiplex: A co-simulation architecture for elevators validation. In *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, pages 1–6. IEEE, 2017.

[San16] Ana B. Sanchez. *Test case prioritization in highly configurable systems.* phdthesis, Universidad de Sevilla, May 2016.

[SCC16] Hema Srikanth, Mikaela Cashman, and Myra B Cohen. Test case prioritization of build acceptance tests for an enterprise cloud application: An industrial case study. *Journal of Systems and Software*, 119:122–135, 2016.

[SEA+17] Goiuria Sagardui, Leire Etxeberria, Joseba A Agirre, Aitor Arrieta, Carlos Fernando Nicolas, and Jose Maria Martin. A configurable validation environment for refactored embedded software: An application to the vertical transport domain. In *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 16–19. IEEE, 2017.

[SER21] Qunying Song, Emelie Engström, and Per Runeson. Concepts in testing of autonomous systems: Academic literature and industry practice. *arXiv preprint arXiv:2103.07157*, 2021.

[SF12] Sriram Sankaranarayanan and Georgios Fainekos. Simulating insulin infusion pump risks by in-silico modeling of the insulin-glucose regulatory system. In *International Conference on Computational Methods in Systems Biology*, pages 322–341. Springer, 2012.

[SGAK15] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-influence models for highly

configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 284–294, 2015.

[SGMM17] Helge Spieker, Arnaud Gotlieb, Dusica Marijan, and Morten Mossige. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 12–22. ACM, 2017.

[SGS+15] Atri Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. Cost-efficient sampling for performance prediction of configurable systems (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 342–352. IEEE, 2015.

[SH09] H. Shokry and M. Hinchey. Model-based verification of embedded software. *Computer*, 42(4):53 – 59, 2009.

[SJH17] Muhammad Sahak, Dayang NA Jawawi, and Shahliza A Halim. An experiment of different similarity measures on test case prioritization for software product lines. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 9(3-4):177–185, 2017.

[SKN+11] Detlef Streitferdt, Florian Kantz, Philipp Nenninger, Thomas Ruschival, Holger Kaul, Thomas Bauer, Tanvir Hussain, and Robert Eschbach. *Model-Based Testing of Highly Configurable Embedded Systems in the Automation Domain*, volume 2, chapter chapter 10, pages 22–41. IGI Global, 2011.

[SKT+16] Reimar Schröter, Sebastian Krieter, Thomas Thüm, Fabian Benduhn, and Gunter Saake. Feature-model interfaces: the highway to compositional analyses of highly-configurable systems. In *Proceedings of the 38th International Conference on Software Engineering*, pages 667–678, 2016.

[SL12] S. Shyam Sunder and Edward A. Lee. Cyber-physical systems - a concept map. techreport, UC Berkeley, December 2012.

[SLB+11] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wąsowski, and Krzysztof Czarnecki. Reverse engineering feature models. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 461–470, 2011.

[SLY+21] Safdar Safdar, Hong Lu, Tao Yue, Shaukat Ali, and Kunming Nie. A framework for automated multi-stage and multi-step product configuration of cyber-physical systems. *Software and Systems Modeling*, 19, 02 2021.

[SMP10] Vanessa Stricker, Andreas Metzger, and Klaus Pohl. Avoiding redundant testing in application engineering. In *Software Product Lines: Going Beyond*, pages 226–240. Springer, 2010.

[SNS+18] Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel C. Briand, and Frank Zimmer. Test case prioritization for acceptance testing of cyber physical systems: A multi-objective search-based approach. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2018, pages 49–60, New York, NY, USA, 2018. ACM.

[SRG11] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. A comparison of decision modeling approaches in product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 119–126, New York, NY, USA, 2011.

[SRJ+17] Houbing Song, Danda B Rawat, Sabina Jeschke, et al. *Cyber-physical systems: Foundations, principles and applications*. Academic Press/Elsevier, 2017.

[SSPRC15] Ana B Sánchez, Sergio Segura, José A Parejo, and Antonio Ruiz-Cortés. Variability testing in the wild: the drupal case study. *Software & Systems Modeling*, 16(1):173–194, 2015.

[SSRC14a] Ana B. Sánchez, S. Segura, and Antonio Ruiz-Cortés. A comparison of test case prioritization criteria for software product lines. In *IEEE International Conference on Software Testing, Verification, and Validation*, pages 41–50, 2014.

[SSRC14b] Ana B Sánchez, Sergio Segura, and Antonio Ruiz-Cortés. The drupal framework: A case study to evaluate variability testing techniques. In *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, page 11. ACM, 2014.

[SWYS11] J. Shi, J. Wan, H. Yan, and H. Suo. A survey of cyber-physical systems. In *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, Nov 2011.

[TAK+14] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys (CSUR)*, 47(1):6, 2014.

[THHB14] Stephen W. Thomas, Hadi Hemmati, Ahmed E. Hassan, and Dorothea Blostein. Static test case prioritization using topic models. *Empirical Software Engineering*, 19(1):182–212, 2014.

[TKB+14] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 79:70 – 85, 2014. Experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010).

[TTK04] Antti Tevanlinna, Juha Taina, and Raine Kauppinen. Product family testing: a survey. *ACM SIGSOFT Software Engineering Notes*, 29(2):12–12, 2004.

[UKB10] E. Uzuncaova, S. Khurshid, and D. Batory. Incremental test generation for software product lines. *IEEE Transactions on Software Engineering*, 36(3):309–322, May 2010.

[UL10] M. Utting and B. Legeard. *Practical model-based testing: a tools approach.* Elsevier, 2010.

[UPL12] M. Utting, A. Pretschner, and B. Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22:297–312, 2012.

[VAHT+18] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. A classification of product sampling for software product lines. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*, pages 1–13, 2018.

[VG07] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. In *11th International Software Product Line Conference (SPLC 2007)*, pages 233–242. IEEE, 2007.

[VK04] Vijay Vaishnavi and William Kuechler. Design research in information systems. Published on design science research in information systems and technology website http://desrist.org/desrist/, January 2004.

[WAC12] James A Whittaker, Jason Arbon, and Jeff Carollo. *How Google tests software.* Addison-Wesley, 2012.

[WAG13] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. In *Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, pages 1493 – 1500, Amsterdam, Netherlands, 2013.

[WAG15] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software*, 103:370–391, may 2015.

[WAGL15] Shuai Wang, Shaukat Ali, Arnaud Gotlieb, and Marius Liaaen. Automated product line test case selection: industrial case study and controlled experiment. *Software & Systems Modeling*, pages 1–25, 2015.

[WAGL16a] Shuai Wang, Shaukat Ali, Arnaud Gotlieb, and Marius Liaaen. A systematic test case selection methodology for product lines: results and insights from an industrial case study. *Empirical Software Engineering*, pages 1–37, 2016.

[WAGL16b] Shuai Wang, Shaukat Ali, Arnaud Gotlieb, and Marius Liaaen. A systematic test case selection methodology for product lines: results and insights from an industrial case study. *Empirical Software Engineering*, 21(4):1586–1622, 2016.

[WAY+16] Shuai Wang, Shaukat Ali, Tao Yue, Yan Li, and Marius Liaaen. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, pages 631–642. ACM, 2016.

[WBA+14] Shuai Wang, David Buchmann, Shaukat Ali, Arnaud Gotlieb, Dipesh Pradhan, and Marius Liaaen. Multi-objective test prioritization in software product line testing: An industrial case study. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 32–41, New York, NY, USA, 2014. ACM.

[Wei08] David M. Weiss. The product line hall of fame. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 395–, Washington, DC, USA, 2008. IEEE Computer Society.

[WGAL13] Shuai Wang, Arnaud Gotlieb, Shaukat Ali, and Marius Liaaen. Automated test case selection using feature model: An industrial case study. In Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke, editors, *Model-Driven Engineering Languages and Systems: 16th International Conference, MODELS 2013, Miami, FL, USA,*

September 29 – October 4, 2013. Proceedings, pages 237–253, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[WL13] S. Weißleder and H. Lackner. Top-down and bottom-up approach for model-based testing of product lines. In *MBT*, pages 82–94, 2013.

[WNK14] H. Wu, C. Nie, and F.-C. Kuo. Test suite prioritization by switching cost. In *IWCT '14*, 2014.

[WSKR06] Kristen R. Walcott, Mary Lou Soffa, Gregory M. Kapfhammer, and Robert S. Roos. Time-aware test suite prioritization. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, ISSTA '06, pages 1–12, New York, NY, USA, 2006. ACM.

[XXL18] Li Da Xu, Eric L Xu, and Ling Li. Industry 4.0: state of the art and future trends. *International Journal of Production Research*, 56(8):2941–2962, 2018.

[YH07] S. Yoo and M. Harman. Pareto efficient multi-objective test case selection. In *ISSTA '07*, pages 140–150, 2007.

[YH12] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22:67–120, 2012.

[YHTS09] Shin Yoo, Mark Harman, Paolo Tonella, and Angelo Susi. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ISSTA '09, pages 201–212, New York, NY, USA, 2009. ACM.

[YSN+20] Jean-Paul A Yaacoub, Ola Salman, Hassan N Noura, Nesrine Kaaniche, Ali Chehab, and Mohamad Malli. Cyber-physical systems security: Limitations, issues and future trends. *Microprocessors and Microsystems*, 77:103201, 2020.

[ZGHY18] Xin Zhou, Xiaodong Gou, Tingting Huang, and Shunkun Yang. Review on testing of cyber physical systems: Methods and testbeds. *IEEE Access*, 6:52179–52194, 2018.

[ZHZ+13] Lingming Zhang, Dan Hao, Lu Zhang, Gregg Rothermel, and Hong Mei. Bridging the gap between the total and additional test-case prioritization strategies. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 192–201, Piscataway, NJ, USA, 2013. IEEE Press.

[ZN08] Justyna Zander-Nowicka. *Model-based testing of real-time embedded systems in the automotive domain.* phdthesis, Electrical Engineering and Computer Science, Technical University Berlin, December 2008.

[ZSM17] Justyna Zander, Ina Schieferdecker, and Pieter J Mosterman. *Model-based testing for embedded systems.* CRC press, 2017.

[ZSV18] Qi Zhu and Alberto Sangiovanni-Vincentelli. Codesign methodologies and tools for cyber–physical systems. *Proceedings of the IEEE*, 106(9):1484–1500, 2018.