Mondragon
Unibertsitatea

DOCTORAL THESIS

SIMULATION-BASED METAMORPHIC TESTING OF CYBER-PHYSICAL SYSTEMS

JON AYERDI CANTALEJO | Arrasate-Mondragón, 2022

JON AYERDI CANTALEJO | Simulation-based Metamorphic Testing of Cyber-Physical Systems

Mondragon Unibertsitatea Goi Eskola Politeknikoa

PHD THESIS

Simulation-based Metamorphic Testing of Cyber-Physical Systems

Author:
JON AYERDI CANTALEJO

Supervisors:
Dr. AITOR ARRIETA MARCOS
Dr. GOIURIA SAGARDUI MENDIETA

Computer and Electronics Department
Faculty of Engineering
Mondragon Unibertsitatea

Arrasate
October 2022

“A wise man does not pretend to know what he know not. He will listen to the wiser”
Terry a O’Neal

Dedicada a mis padres,
Jon

Declaration

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

Jon Ayerdi Cantalejo
Arrasate, October 2022

Acknowledgments

I would first like to thank my supervisors Aitor and Goiuria for guiding me throughout this journey. Both of them have been supportive and helpful whenever I faced challenges, and this thesis could not have been completed without their advice. I thank Aitor for getting me started on research while I was still a masters student and for giving me the opportunity to do this PhD. He has given me the freedom to work on whichever research topics I was interested in while also guiding me when I was lost. His strict supervision has pushed me to make progress, and eventually succeed at finishing this thesis within 3 years, having no prior research experience.

I also thank my colleagues from my research group for their continuous support. Thanks to Leire, Miren and Miriam for their advice and support during this PhD. Special thanks to Urtzi for reaching out to help in times of need.

I would also like to thank Sergio Segura for his guidance and help, and for welcoming me at his research group at the University of Seville. His advice has been essential, and has greatly shaped the direction of this thesis from its early stages.

I thank Prof. Paolo Tonella and his group for welcoming me at their research group at Università della Svizzera italiana (USI). Special thanks to Valerio Terragni, Gunel Jahangirova and Paolo Tonella for working with me, and for their efforts during difficult times.

Gracias a mis compañeros, a mi familia, y especialmente a mis padres por haberme apoyado siempre hasta ahora.

I thank the Software and Systems Engineering research group of Mondragon Goi Eskola Politeknikoa for their financial support. This thesis has been partially supported by a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871319. Jon Ayerdi, Aitor Arrieta and Goiuria Sagardui are part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1519-22).

Abstract

Cyber-Physical Systems (CPSs) are heterogeneous systems in which software components and physical processes interact with each other. These systems have strict requirements that are difficult to verify, resulting in very costly verification processes which often lack automation. As CPSs and their requirements become more complex, development practices that still rely on manual intervention struggle to scale up. As such, there is a need to identify the fundamental problems that currently prevent the automation of those processes, and find solutions to them. One such problem is the inability to accurately define the expected behaviour of the systems for their verification, namely, the test oracle problem. Due to factors such as ambiguous requirements or non-determinism, testing activities for CPSs often involve human oracles, since automated oracles cannot be implemented without defining specific properties to verify.

Metamorphic Testing (MT) is an alternative verification approach which can be used to define test oracles. Instead of focusing on a single system execution, MT defines properties that compare the behaviours from two or more system executions with known relationships between their inputs and outputs, the so-called Metamorphic Relations (MRs).

This thesis aims to advance the current practice on testing CPSs by proposing methods that alleviate the test oracle problem by using MT. First, we present a general CPS DevOps architecture which will enable the automation of all the development tasks, focusing on the verification components where the MT techniques will be implemented. Second, we propose methods for identifying MRs for CPSs. On the one hand, we propose a pattern for performance-based MRs, which can be used to instantiate MRs for specific CPSs more easily. On the other hand, we implement a fully-automated approach for generating MRs for a given system based on samples of correct and incorrect system behaviours. We implement two prototypes of this technique. The first prototype enables automatic performance MR generation for CPSs, given user-provided test execution pairs. The second prototype demonstrates the full automation of a generic whole MR generation process. Finally, we propose an

optimization approach for MT, metamorphic test selection. The goal of this approach is to maximize the cost-effectiveness of MT by reducing the number of tests that need to be executed, which is important due to the high cost of executing CPSs.

Laburpena

Sistema ziber-fisikoak (CPSak) software-osagaiek eta prozesu fisikoek elkarri eragiten dioten sistemak dira. Sistema hauek balidatzeko zailak diren propietateak izan ohi dituzte, eta ondorioz sarritan automatizaziorik ez duten balidazio prozesuak izaten dituzte. CPSak eta heuren propietateak konplexuagoak bihurtzen diren heinean, interbentzio manualetan oinarritzen diren praktikek eskalatzeko harazoak izaten dituzte. Hori dela eta, prozesu horien automatizazioa eragozten duten oinarrizko arazoak identifikatu beharra dago, eta horiei irtenbideak bilatu. Arazo horietako bat sistemaren balidaziorako espero den portaera zehaztasunez definitzeko ezintasuna da, hau da, orakuluaren arazoa. Baldintza anbiguoak eta ez-determinismoa bezalako faktoreak direla eta, CPSetarako balidazioek askotan orakulu manualak izaten dituzte, orakulu automatizatuak ezin baitira inplementatu balidatzeko propietate zehatzik gabe.

Metamorphic Testing (MT) orakuluak definitzeko erabil daitekeen balidazio teknika alternatibo bat da. Sistemaren exekuzio bakarra erabili beharrean, MTek bi exekuzio edo gehiagoren portaerak alderatzen dituzten propietateak definitzen ditu, haien sarrera eta irteeren arteko erlazio ezagunak dituztenak, erlazio metamorfikoak (MRs) deiturikoak.

Tesi honen helburua CPSak balidatzeko egungo jardunbideak hobetzea da, MT teknika erabiliz orakuluaren arazoari irtenbidea hematen dioten metodoak proposatuz. Lehenik eta behin, sistemaren garapena automatizatzea ahalbidetuko duen CPS DevOps arkitektura orokor bat aurkezten dugu, MT teknikak implementatuko dituzten balidazio osagaietan zentratuz. Bigarrenik, CPSetarako MRak identifikatzeko metodoak proposatzen ditugu. Alde batetik, errendimendu metriketan oinarritutako MRetarako txantiloi bat proposatzen dugu, CPSetarako MRak errazago instantziatzeko erabil daitekeena. Bestalde, sistema baterako MRak sortzeko teknika automatizatu bat proposatzen dugu, sistemaren jokabide zuzen eta okerren laginetan oinarrituta. Teknika honen bi prototipo inplementatzen ditugu. Lehenengo prototipoak errendimendu metriketan oinarritutako MRen sorkuntza ahalbidetzen du CPSetarako, erabiltzaileek emandako test bikoteak erabiliz. Bigarren prototipoak MRak sortzeko prozesuaren automatizazio osoa demostratzen du. Azkenik, MTrako optimizazio bat proposatzen dugu, test metamorfikoen selekzioa. Teknika honen helburua exekutatu behar diren test kopurua murriztuz MTen eraginkortasuna handitzea da, eta hori garrantzitsua da CPSak exekutatzeko kostu handia dela eta.

Resumen

Los sistemas cyber-físicos (CPSs) son sistemas heterogéneos en los que componentes de software y procesos físicos interactuan entre ellos. Estos sistemas tienen requisitos estrictos que son difíciles de verificar, lo que resulta en costosos procesos de verificación que no suelen estar automatizados. A medida que los CPSs y sus requisitos se vuelven más complejos, las prácticas del proceso de desarrollo que dependen de la intervención manual escalan cada vez peor. Debido a esto, hay una necesidad de identificar los problemas fundamentales que bloquean la automatización de estos procesos, así como de encontrar soluciones a estos problemas. Uno de estos problemas es la dificultad de definir el comportamiento esperado de los sistemas durante su verificación, lo que se conoce como el problema del oráculo. Debido a factores como los requisitos ambíguos o el no determinismo de los sistemas, las actividades de testeo para CPSs suelen involucrar oráculos humanos, puesto que la implementación de oráculos de testeo automatizados requiere la definición de propiedades específicas a verificar.

Metamorphic Testing (MT) es una técnica de verificación alternativa que permite definir oráculos de testeo. El lugar de centrarse en ejecuciones individuales del sistema, MT permite definir propiedades basadas en múltiples ejecuciones con relaciones específicas entre las entradas y salidas del sistema, las llamadas relaciones metamórficas (MRs).

Esta tesis tiene como objetivo avanzar en la práctica actual del testeo de sistemas cyber-físicos mediante la propuesta de métodos para aliviar el problema del oráculo de testeo mediante el uso de MT. En primer lugar, presentamos una arquitectura DevOps para CPSs que permitirá automatizar todas las tareas de desarrollo del sistema, centrandonos en los componentes de verificación en los que se integrarán las técnicas de MT. En segundo lugar, proponemos métodos para identificar MRs en CPSs. Por una parte, proponemos un patrón general para MRs basadas en métricas de rendimiento, el cual puede ayudar a instanciar MRs para CPSs concretos de forma más fácil. Por otra parte, implementamos una técnica completamente automática para generar MRs para cualquier sistema a partir de ejecuciones que demuestren su comportamiento correcto e incorrecto. Desarrollamos dos prototipos de esta técnica. El primer prototipo es capaz de generar MRs basadas en métricas de rendimiento de forma automática a partir de parejas de ejecuciones del sistema proporcionadas por los usuarios. El segundo prototipo demuestra la automatización completa del proceso de generación de MRs genéricas. Finalmente, proponemos una técnica de optimización para MT que consiste en la selección de testeos metamórficos. La finalidad de esta técnica es maximizar la eficiencia de MT mediante la reducción del número de testeos que tienen que ser ejecutados, lo cual es muy relevante dado el alto coste de ejecutar CPSs.

Contents

List of Figures

List of Tables

Acronyms

ADS	Autonomous Driving System
ADT	Abstract Data Type
AMT	Automatic Metamorphic Testing
ANN	Artificial Neural Network
API	Application Programming Interface
ASM	Abstract State Machine
AWT	Average Waiting Time
CAN	Controller Area Network
CGC	Conventional Group Control
CPPS	Cyber-Physical Production System
CPS	Cyber-Physical System
CSE	Continuous Software Engineering
DM	Decision Maker
DNN	Deep Neural Network
DSL	Domain-Specific Language
DSR	Design Science Research
FDC	Fault Detection Capability
FDR	Failure Detection Ratio
FP	False Positive

FSM	Finite State Machine
GA	Genetic Algorithm
GAN	Generative Adversarial Network
GP	Genetic Programming
HDL	Hardware Description Language
HiL	Hardware-in-the-Loop
HV	Hypervolume
IoT	Internet of Things
JML	Java Modelling Language
LTL	Linear Temporal Logic
MBD	Model-Based Design
MBT	Model-Based Testing
MiL	Model-in-the-Loop
ML	Machine Learning
MR	Metamorphic Relation
MRIP	Metamorphic Relation Input Pattern
MROP	Metamorphic Relation Output Pattern
MRP	Metamorphic Relation Pattern
MS	Mutation Score
MSE	Mean Squared Error
MT	Metamorphic Testing
PiL	Processor-in-the-Loop
PSO	Particle Swarm Optimization
PV	Performance Variation
QA	Quality Assurance

QoS	Quality of Service
RFOL	Restricted Signals First-Order Logic
RT	Real-Time
RQ	Research Question
SBO	Search-Based Optimization
SBSE	Search-Based Software Engineering
SiL	Software-in-the-Loop
SMC	Statistical Model Checking
SMT	Satisfiability Modulo Theory
SUT	System Under Test
SVM	Support Vector Machine
TCMS	Train Control and Management System
TD	Total Distance
TM	Total Movements
TTD	Time To Destination
TTO	Total Trajectory Offset
UML	Unified Modeling Language

Part I

Foundation and Context

Chapter 1

Introduction

This chapter introduces the main motivation and scope of the research work carried out by the Ph.D. candidate, and the problems that have been tackled. The research methodology is presented, and the main technical contributions and the corresponding publications are highlighted. Other related activities, such as research stays and research project contributions, are also described.

1.1 Motivation and Scope

The term Cyber-Physical System (CPS) refers to heterogeneous systems of networked computational entities and software that interact with the physical world [Lee08]. This paradigm is a holistic approach to systems development where different engineering disciplines are deeply intertwined with each other. The applications of CPSs extend to many areas, such as aerospace, automotive, healthcare, manufacturing and consumer appliances [Lee08].

Many of these applications have strict safety requirements, whereas others require the products to be resilient to failures while interacting with humans or operating in unknown environments [KM14]. Because of these requirements, verification is one of the major concerns when it comes to CPS development. In several application domains, the verification process requires over 50% of the development resources [BG11]. This is partly due to the high complexity of these systems, as well as the inherent uncertainty of their interactions with the physical environment, which can make generating test oracles for them infeasible [KFK14].

This is a particularly common issue when non-functional properties such as performance indicators [NJT13] or Quality of Service (QoS) metrics [CP13] are involved, since there are often no mechanisms to determine if the values obtained for such metrics are correct.

For instance, multi-elevator installations developed by Orona, one of the leading elevator companies in Europe, suffers from this problem. In a building with multiple elevators attending calls, it is difficult to estimate the expected values of QoS metrics (e.g. passenger waiting times, energy consumption...), and it is therefore very hard to identify failures which are affecting these metrics negatively (e.g. caused by a software bug in the dispatching algorithm) [ASA+20]. Some hard requirements can be defined, such as a general maximum waiting time for the passengers, but such rigid requirements must necessarily be permissive, and are therefore only useful to identify the most blatant failures. Regression oracles are currently the only form of fully automated testing widely used by Orona, since they allow the identification of failures that result in a worse QoS than a previous software release. Nevertheless, such oracles are only applicable when an appropriate baseline is available. For example, a system that must be deployed to a new building will still require extensive manual verification, since the data from a different building is not comparable.

This inability to define test oracles due to the lack of an exact specification of properties for the System Under Test (SUT) or the high difficulty of verifying known properties is referred to as the test oracle problem [BHM+14]. In such cases, testing and verification processes can still be performed by human oracles, but manual processes result in a much higher cost compared with fully automated ones.

The high complexity of these systems also results in very costly configuration, deployment and execution tasks, which often involves some manual intervention in practice [AGA+20]. Due to this reliance on human involvement, the release cycles for new software versions are often very long, which makes keeping up with new regulations, feature requests and bug fixes increasingly difficult.

In contrast to this, other domains such as web engineering have already adopted methods such as DevOps [EGHS16], which streamline the software development process by automating the transition from a commit in the source repository to the verification and deployment to production, and also automate the monitoring and verification of the deployed system during its operation. A DevOps approach could greatly reduce the duration of release cycles for CPSs. However, adopting it in this context requires the definition and implementation of automated approaches for all the related tasks, including deployment, monitoring, and verification. In order to accomplish that, fundamental issues such as the test oracle problem also need to be addressed to the point where tool-supported full automation becomes feasible.

Regarding the oracle problem, Metamorphic Testing (MT) is a technique which can alleviate this issue by defining an alternative type of system properties. Instead of defining properties over individual system executions, MT uses known input and output relations that should hold among multiple executions, the so-called Metamorphic Relation (MR) [CCY98]. In multi-elevator installations, although predicting the QoS of a given test scenario installation is difficult, we can define changes to a test scenario which should have a predictable impact on the QoS. For example, if the elevators are made faster, the Average Waiting Time (AWT) of the passengers can be expected to decrease or remain the same in any scenario. This way, we can identify system failures from cases where the faster elevators obtain a worse AWT, since this should never happen if the system is working correctly.

Considering the issues described above, we identified the following challenges to tackle for achieving a full automation of the CPS development process:

- Define a DevOps system suitable for CPS development, which must be reliable and flexible enough to address the various concerns and needs from different sub-domains.
  - Propose an approach to *manage the requirements* for a CPS DevOps system. Due to the specific challenges posed by CPSs, these requirements will be different from other domains where DevOps is already in use, such as web engineering. Furthermore, the requirements of a DevOps system might change significantly in different sub-domains.
  - *Design an architecture* which is flexible enough to accommodate the DevOps needs from different CPS developers. This architecture should be adaptable to the different development paradigms and tools that are used for CPSs, and its subsystems should be loosely coupled in order to facilitate their addition or removal.
- Propose an approach to *alleviate the oracle problem*, which is one of the fundamental issues preventing the automation of verification activities [BHM+14]. *MT* is a potentially useful alternative approach which could be employed to tackle this challenge, but more research needs to be done on applying this technique in the domain of CPSs.
  - Generating MRs manually is a task which requires extensive practical experience with the CPS under test. Metamorphic Relation Patterns (MRPs) can aid in this task by providing reusable templates which can be used to instantiate MRs in new domains [SPTRC18, ZSCT18]. The study of such patterns, particularly for the domain of CPSs is a topic which is yet largely unexplored, and this work must be done in order to assess the usefulness of MT in this context.
  - As an alternative to defining MRs manually, automated approaches may be able to identify such properties for a given system. Such techniques must be able to generate MRs that are *valid* (i.e. have no false positives), but also *effective* (i.e.

as few false negatives as possible) and, ideally, not excessively *complex* for a human to understand.

- Propose techniques for *optimizing MT* in the context of CPS verification. Testing CPSs is usually a costly process, and as such, running a full test suite is often not possible [BG11]. Furthermore, MRs require multiple test executions to check, further increasing the testing costs with respect to other techniques. Test case selection is one of the most basic optimization techniques, and consists in selecting a subset of the test suite in order to maximize its cost-effectiveness [YH12]. Although many related techniques exist, an assessment of the most efficient algorithms and fitness functions needs to be made for the context of CPSs.

1.2 Research Methodology

The research method of this Ph.D. followed the guidelines for the Design Science Research (DSR) paradigm, as described by Hevner and Chatterjee [HC10].

Hevner and Chatterjee identified three different cycles that are present in any design research project [HC10]. On the one hand, the *Relevance Cycle* bridges the contextual environment with the research activities. On the other hand, the *Rigor Cycle* provides the existing knowledge base to the research project in order to ensure the innovativeness of the contributions that are made. Finally, the *Design Cycle* is the heart of the project, where new designs are generated and evaluated until a satisfactory result is obtained. This thesis, as well as each of its individual contributions, have followed the guidelines related to each of these cycles [HC10].

Our research followed the DSR guidelines detailed in [HMPR04].

- **Guideline 1: Design as an artifact**. We produce viable artifacts for each of the contributions of this thesis, mostly in the form of methods and tools.

- **Guideline 2: Problem relevance**. Our contributions solve relevant problems, as they are motivated by industrial case studies.

- **Guideline 3: Design evaluation**. We attempt to demonstrate the effectiveness of our solutions, mostly by performing experimental evaluations on industrial case studies.

- **Guideline 4: Research contributions**. Verifiable research contributions to the state of the art on the relevant topics are made for each part of this thesis.

- **Guideline 5: Research rigor**. For each contribution, we followed rigorous development and evaluation processes, mostly based on the standards set by notable research works on the related topics.
- **Guideline 6: Design as a search process**. We followed an iterative process to develop designs that efficiently solve the problems we set to solve, evaluating and refining possible solutions until satisfactory results were achieved.
- **Guideline 7: Communication of the research**. Our research is presented in terms of both technological advancements and tangible costs and benefits for the adopting organizations.

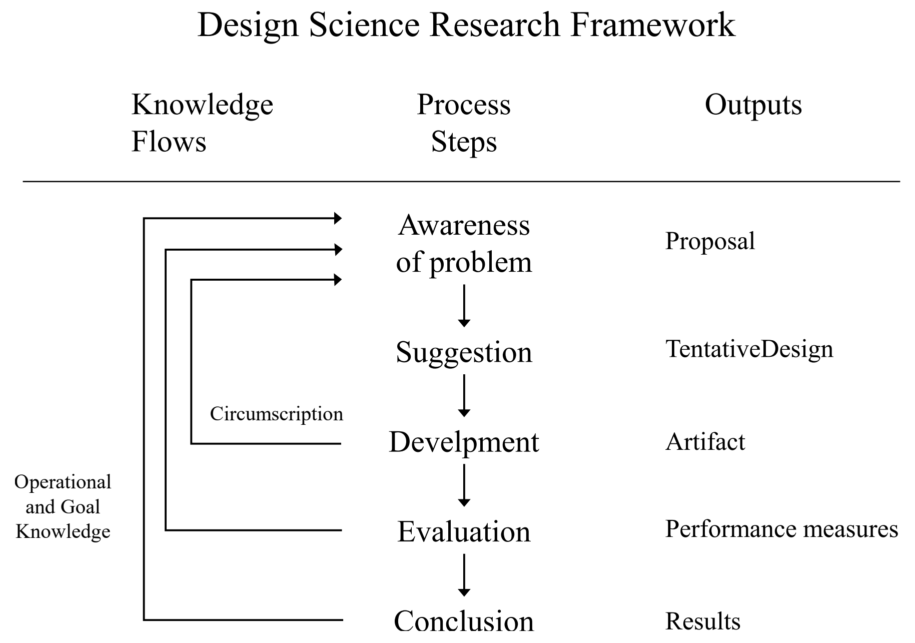


Figure 1.1: DSR process model [VK15]

The specific development process followed the general model described by Vaishnavi and Kuechler [VK15]. This process model is shown in Figure 1.1, and comprises the following steps:

- **Awareness of the problem**. The awareness of an existing research problem may come from sources such as new developments in industry. The output of this process is a formal or informal proposal for a new research effort.
- **Suggestion**. This is a creative step where new functionality is envisioned, and a tentative design is suggested. This design is usually an integral part of a research proposal.

- **Development**. The tentative design is further developed and implemented, and a novel artifact is created. The novelty of the artifact usually resides solely in its design.

- **Evaluation**. The developed artifact is evaluated based on criteria which are often made explicit in the proposal. This stage contains an analytic sub-phase where, for every deviation from the expected results that is detected, hypotheses are made to explain the behaviour, the hypotheses are tested and adjusted, and the resulting knowledge is fed back to the previous stages.

- **Conclusion**. The research cycle ends when the requirements specified in the previous steps are met. The results from the process are consolidated and the obtained knowledge is disseminated.

1.3 Research Contributions

The main contributions of this thesis can be summarized as follows:

- A taxonomy for eliciting requirements for CPS DevOps systems is presented. Our taxonomy considers four different dimensions of the requirements: Scope, domain, lifecycle stage, and subsystem. The presented taxonomy is a template which can be extended and adapted depending on the needs of the practitioners from different domains. This work is based on two industrial case studies from the elevation and railway domains, for which we elicit requirements in order to evaluate and iterate our taxonomy. Based on their needs, our taxonomy comprises DevOps subsystems for: Deployment, monitoring, validation and integration. Our work for this thesis has mainly focused on the validation subsystem. This contribution has been published in the IEEE 28th International Requirements Engineering Conference (RE) [AGA$^+$20].

- An architecture for CPS DevOps systems based on microservices is defined. This architecture follows the taxonomy and requirements elicited in our previous contribution, and instantiates a solution for the case study from the elevation domain. Our solution defines microservices which implement the various subsystems, which employ both synchronous (REST) and asynchronous (MQTT) communications to integrate with each other. The subsystems defined include deployment, monitoring and validation, each of which comprises a server-side orchestrator and agents which perform operations directly on the target nodes (i.e. the CPSs). Furthermore, an automation server integrates all of these subsystems by triggering the DevOps pipeline when a new plan is launched (e.g. when a code commit is made). This involves generating the artifacts for each subsystem, which are pulled from a Docker registry, and coordinating their respective actions. Once again, the contribution of this thesis lies mainly in the validation subsystem of the architecture. This contribution has been published in the IEEE 28th International Requirements Engineering Conference (RE) [GAA+21b].

- A method for generating performance MRs based on patterns is proposed. The core contribution is the proposed Performance Variation (PV) MRP, a general pattern which represents MRs that involve input changes that have a predictable impact on performance measurements. We exploit this pattern in our case study from the elevation domain, deriving various Metamorphic Relation Input Patterns (MRIPs) from the PV pattern, and then defining MRs based on the proposed MRIPs and domain-specific performance metrics. In our experimental evaluation we show that, as a solution to alleviate the test oracle problem, the MRs we develop outperform regression and threshold-based oracles, which are the only existing alternative solutions for our case studies. An initial version of this contribution has been published in the IEEE 31st International Symposium on Software Reliability Engineering (ISSRE) [ASA+20], and an extension introducing a new open-source case study from the autonomous driving domain has also been published in the IEEE Transactions on Reliability journal [AVS+22a]. A replication package for the open-source case study experiments has also been included in the latter publication [AVS+22b].

- A tool-supported automated approach for generating MRs is presented. The approach adapts a previous evolutionary approach for generating Java assertions [TJTP20] to the context of generating performance MRs for CPSs. This evolutionary approach employs correct and incorrect execution traces and a genetic algorithm in order to generate test oracles which can distinguish between them with the fewest false positives and false negatives as possible. An initial stage of this approach generates output relations for some given MRIPs, which can be manually defined, for instance, based on existing MR patterns. We evaluate this approach in the context of our industrial elevation case study, and the results show that the automated approach matches or outperforms the effectiveness of the manually generated MRs. The second stage of this contribution introduces various approaches for generating full MRs from scratch. We implement a tool which can generate MRs for Java methods in a fully automated way by integrating existing mutation testing and test suite generation tools. Our evaluation reveals that this approach is capable of generating valid and effective MRs for 8 out of 10 methods. The initial stage of this contribution has been published in the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) [ATA+21]. An additional publication intended to be submitted to the IEEE International Conference on Software Testing, Verification and Validation (ICST) research track is being worked on, and an extension of this work is also planned to be submitted to the ACM Transactions on Software Engineering and Methodology (TOSEM) journal.

- A search-based *metamorphic test selection* approach for CPSs which optimizes the *cost-effectiveness* of the test suites is proposed. The test suite execution cost is combined with five other objective functions, two of which are generic and three of them specific to the elevation domain. Our approach employs the Non-dominated Sorting Genetic Algorithm II (NSGA-II), but also comprises a problem representation specific to the problem and novel mutation and crossover operators specific to the metamorphic test selection problem. Our approach is evaluated with our case study from the elevation domain, using the randomly generated test suite, MRs and mutants employed in our previous work on manually generating QoS-based MRs [ASA+20]. This contribution has been accepted for publication in the IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE 2022) [AAPA22a]. This publication includes a replication package containing the source code for the metamorphic test case selection implementation, the experimental data, and the results obtained [AAPA22b].

1.4 Publications

Several peer-reviewed journal and conference publications have been achieved during the Ph.D. The journal publications have been rated based on the Journal Citation Reports (JCR), showing their score and quartile. Conference publications show their GII-GRIN-SCIE (GGS)[1] rating, which is the metric endorsed by the Spanish Informatics Scientific Society (SCIE)[2].

1.4.1 Journal Publications

An article was published at the IEEE Transactions on Reliability (TR) journal. Furthermore, at the time this thesis is being submitted, a journal article to be submitted to

[1] http://gii-grin-scie-rating.scie.es/
[2] https://www.scie.es/

the ACM Transactions on Software Engineering and Methodology (TOSEM) is being worked on.

- Aitor Gartziandia, Aitor Arrieta, Jon Ayerdi, Miren Illarramendi, Aitor Agirre, Goiuria Sagardui, Maite Arratibel. "Machine learning-based test oracles for performance testing of cyber-physical systems: An industrial case study on elevators dispatching algorithms" in Journal of Software: Evolution and Process (Special Issue - Methodology Paper), 25 May 2022, pp. e2465. DOI: https://doi.org/10.1002/smr.2465. **JCR: 1.844**. **Q3**.

- Jon Ayerdi, Pablo Valle, Sergio Segura, Aitor Arrieta, Goiuria Sagardui, Maite Arratibel. "Performance-Driven Metamorphic Testing of Cyber-Physical Systems" in IEEE Transactions on Reliability (Early Access), 04 August 2022, pp. 1-19. DOI: https://doi.org/10.1109/TR.2022.3193070. **JCR: 5.15**. **Q1**.

1.4.2 International Conferences

A total of 7 publications were achieved at international conferences. At the time this thesis is being submitted, the ISSRE 2022 conference has not occurred yet, and its proceedings have not been published. Furthermore, an additional article to be submitted to the IEEE International Conference on Software Testing, Verification and Validation (ICST) research track is being worked on.

- Jon Ayerdi, Aitor Garciandia, Aitor Arrieta, Wasif Afzal, Eduard Enoiu, Aitor Agirre, Goiuria Sagardui, Maite Arratibel, Ola Sellin. "Towards a Taxonomy for Eliciting Design-Operation Continuum Requirements of Cyber-Physical Systems" in IEEE 28th International Requirements Engineering Conference (RE), Industrial Innovation track, 09 October 2020. **GGS: A**

- Jon Ayerdi, Sergio Segura, Aitor Arrieta, Goiuria Sagardui, Maite Arratibel. "QoS-aware Metamorphic Testing: An Elevation Case Study" in IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Research Track, 11 November 2020. **GGS: B**

- Aitor Gartziandia, Jon Ayerdi, Aitor Arrieta, Shaukat Ali, Tao Yue, Aitor Agirre, Goiuria Sagardui, Maite Arratibel. "Microservices for Continuous Deployment, Monitoring and Validation in Cyber-Physical Systems: an Industrial Case Study for Elevators Systems" in IEEE 18th International Conference on Software Architecture (ICSA), Software Architecture in Practice Track, 10 May 2021. **GGS: A-**

- Aitor Arrieta, Jon Ayerdi, Miren Illarramendi, Aitor Agirre, Goiuria Sagardui, Maite Arratibel. "Using Machine Learning to Build Test Oracles: an Industrial Case Study on Elevators Dispatching Algorithms" in IEEE/ACM International Conference on Automation of Software Test (AST), 28 June 2021. **GGS: Work in Progress**

- Jon Ayerdi, Valerio Terragni, Aitor Arrieta, Paolo Tonella, Goiuria Sagardui, Maite Arratibel. "Generating metamorphic relations for cyber-physical systems with genetic programming: an industrial case study" in 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Industry Track, 18 August 2021. **GGS: A+**

- Jon Ayerdi, Aitor Arrieta, Ernest Bota Pobee, Maite Arratibel. "Multi-Objective Metamorphic Test Case Selection: an Industrial Case Study" in IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), Research Track, 2022. **GGS: B**

1.5 Related Activities

In addition to developing the mentioned conference and journal publications and attending the corresponding conferences, various other research activities have been performed by the the Ph.D. student. These activities include disseminating the research through talks at a conference and a summer school, two research stays, and participation in an European project.

1.5.1 Talks

The research from this thesis was also disseminated in the Hot off the Press (HoP) track from the GECCO 2022 conference, as well as the SMILESENG summer school.

- Jon Ayerdi, Valerio Terragni, Aitor Arrieta, Paolo Tonella, Goiuria Sagardui, Maite Arratibel. "Evolutionary generation of metamorphic relations for cyber-physical systems" in The Genetic and Evolutionary Computation Conference (GECCO), 2022, Hot off the Press (HoP) track. The talk given by the Ph.D. student disseminated the work carried out on automatic MR generation for CPSs [ATA^{+}21].

- International Summer School on Search and Machine Learning-based Software Engineering (SMILESENG), third edition [3]. The talk given by the Ph.D. student,

[3] https://www.uco.es/smileseng/

entitled “Generating Complex Metamorphic Relations for Cyber-Physical Systems with Genetic Programming”, disseminated the work carried out on automatic MR generation [ATA+21] and proposed some new related research avenues.

1.5.2 Research Stays

During the development of this thesis, the Ph.D. student visited other institutions in order to collaborate with researchers with expertise in particular topics. Two research stays were undertaken, each of which had a duration of three months. The first stay was at the University of Seville, Spain. The second stay was at Università della Svizzera italiana (USI), Switzerland.

- *Universidad de Sevilla, at the Applied Software Engineering (ISA) research group with Prof. Sergio Segura, mid October 2019 to mid December 2019.* The objective of the research stay was to collaborate with expert researchers in the field of MT. During the stay, the Ph.D. student learned about MT and worked towards adapting this technique to CPSs following the feedback from the experts. The results from the collaboration include a conference and a journal publication [ASA+20, AVS+22a].

- *Università della Svizzera italiana (USI), at the TAU research group at the Software Institute with Prof. Paolo Tonella, mid October 2021 to mid December 2021.* The objective of the research stay was to collaborate with expert researchers on the topic of automatic test oracle generation, on which some preliminary work had already been done via remote collaboration [ATA+21]. The results from the collaboration include an article to be submitted to the IEEE International Conference on Software Testing, Verification and Validation (ICST), as well as some preliminary work for a journal extension to be submitted to the ACM Transactions on Software Engineering and Methodology (TOSEM).

1.5.3 Research Projects

The Ph.D. student has been involved in the Adeptness[4] project, which is funded by the European Union’s Horizon 2020 research programme and has the goal of automating the design-operation continuum of CPSs. The research done during the Ph.D. studies had the goal of meeting some of the objectives from this project, so the student contributed to it with several related publications[5], as well as the implementation of some demonstrators for verification-related tasks.

[4] https://adeptness.eu/

[5] https://adeptness.eu/scientific-publications/

1.6 Outline of the Dissertation

This thesis is structured as follows: Part I of the thesis contains the Foundation and Context. Chapter 1 introduces the motivation and scope of this thesis, describes the research methodology, and summarizes the research contributions, publications, and other related activities performed by the Ph.D. student. Chapter 2 presents the basic technical background and terminology used throughout the topics discussed in this thesis. Chapter 3 summarizes the state of the art for the related topics and highlights the potential points for research contributions. Chapter 4 describes the theoretical framework, including the research objectives and hypotheses, an overview of the contributions, and the case studies employed for evaluating the proposed solutions.

Part II corresponds to the context of this thesis, which is the definition of methods for automating CPS development processes. Chapter 5 proposes a taxonomy for eliciting requirements for a DevOps framework for CPSs. Chapter 6 proposes one such DevOps framework, which is based on a microservices architecture.

Part III proposes the use of MT to alleviate the test oracle problem for CPSs. Chapter 7 presents our proposal for the manual definition of MR for CPSs based on specific patterns. Then, Chapter 8 presents a framework for automatically generating MRs.

Part IV addresses the optimization problem for MT. In this regard, Chapter 9 proposes a metamorphic test selection approach for CPSs.

Finally, Part V contains the final remarks of this thesis. In Chapter 10 the contributions that have been made are summarized, and the lessons learned and future prospects are discussed.

Chapter 2

Background

This chapter contains technical background information on various topics within the scope of this thesis. First, Section 2.1 provides the background related to Cyber-Physical Systems (CPSs). Then, Section 2.2 explains the verification techniques used on CPSs. Next, Section 2.3 describes the verification of non-functional aspects of a system. Then, Section 2.4 introduces Metamorphic Testing (MT). Finally, Section 2.5 presents the background on Search-Based Software Engineering (SBSE).

2.1 Cyber-Physical Systems

CPSs are a combination of computation and physical processes that interact with each other in complex ways [Lee08]. They usually consist of feedback loops in which physical processes affect the computations and vice versa. These systems are quickly becoming ubiquitous, and can now be found in various areas, such as avionics, healthcare, or consumer electronics [Lee08].

Due to their interaction with the physical world, the discrete logic of the computing systems must interface with continuous and concurrent dynamics that cannot be fully observed or controlled [KM14]. This introduces uncertainties, which must be contained or tolerated in order to fulfil the requirements of the system [KFK14]. These requirements are not only functional, but also include privacy, security, fault tolerance, and other non-functional requirements [Lee08].

Currently, most of the software related technologies, such as programming languages and concurrency models, are built on abstractions that do not match real-time physical systems correctly. For instance, the semantics of the C programming language do not consider aspects of the behaviour of the program such as the timing of the execution, which makes it unpredictable for real-time applications [Lee08]. Many CPSs also contain device networks, and therefore networking technologies also need to be adapted in order to accomplish a predictable and reliable real-time performance needed for these systems [Lee08].

The following list summarizes the main challenges that CPS engineering faces:

- *Heterogeneous engineering disciplines*: Apart from software engineering, CPSs usually require the collaboration with experts from many other engineering disciplines. The gap between these disciplines and the different formalisms they use needs to be closed in order to integrate all the partial solutions into a product [Par10, GTB$^+$17], and the later this integration is done in the development process, the less optimal it is [TDUE07].

- *Complexity*: On the other hand, the increasing complexity of these systems makes developing them, and especially performing quality assurance, very challenging and time-consuming. This is especially the case when the systems are autonomous, which makes predicting and understanding their behaviour very challenging.

- *Variability*: Product Line Engineering is an approach that improves the reusability of the components for different product variants, thus enhancing the productivity and quality of the development process. CPSs in particular are systems that contain high variability due to the different environments and scenarios in which they operate. This variability introduces additional challenges, especially when it comes to verification, since verifying the entire product line is usually not feasible.

- *Uncertainty*: Another issue that CPSs face compared to purely cyber systems is their stochastic nature, which can make them unpredictable to a certain extent [ZSA$^+$16]. Some of the uncertainty sources in CPSs are:

  - *The unpredictable environment* is a major source of uncertainty for CPSs. Some characteristics of the operational environment, such as different weather conditions, can have a very significant impact on the behaviour of the system and are difficult to predict. Furthermore, the system may be used in a context which was not contemplated in its design, such as different temperature or humidity conditions [dWOJ$^+$07].

  - *Human intervention* both during development and runtime, is also intrinsically inexact and susceptible to errors. CPS development is still largely based on human participation, and the errors introduced to the system can vary in unpredictable ways based on many factors, such as developer experience and team workflow, or the skills of the operators of the product [dWOJ$^+$07].

  - *Unreliable components* are another source of uncertainty which can make the system behave unpredictably. Even when the developers are already familiar with a component, it is not always known how accurate and reliable it will be when used in a new context or interacting with other parts of the system [dWOJ+07].

- *Software* itself does not, in general, have a fully deterministic real-time behaviour, especially if concurrency or communication with other devices are involved [Ram00]. This is even more so the case for machine learning applications or any kind of adaptive systems in general, where the behaviour of the system is uncertain due to the sheer complexity of its decision-making process.

2.2 Verification of Cyber-Physical Systems

The increased use of CPSs brings the need for new Quality Assurance (QA) processes to identify and fix defects in these systems, especially in the case of safety-critical applications. Analogously to the other related technologies, system and software verification techniques and tools need to be adapted to the context of CPSs.

There are many verification techniques that have already been proposed for CPSs. The following list shows an overview of the available methods [ZJKK15]:

- **Formal methods**
  - **Static analysis** is a technique which can provide approximate but sound guarantees about the behaviour of a program without actually executing it [DKW08]. This term encompasses a family of techniques which is typically used by compilers (e.g. for emitting warnings) and their optimizers, but it can also be used for verification purposes. This is accomplished by computing invariants, and thus proving properties of the program. These approaches have also been applied to CPSs, but they tend to not scale well or introduce many false positives as the programs get larger and have more complex interactions [ZJKK15].
  - **Theorem proving** consists in translating a given program into a logical formula whose validity reflects the correctness of the program, and then passing that formula to a theorem prover which checks the formula. For non-trivial programs, this involves significant human intervention and various refinement passes from an abstract specification of the program to the source code implementation. In [KEH+09], an operating system kernel which is part of a complex CPS is fully verified using this technique, although it required a careful design of the system, and the effort for proving its correctness was about 20 person-years (and this proof assumes the correctness of the compiler, hardware...etc.).
  - **Model Checking** is a technique for checking whether a given model meets a given specification in an automated and exhaustive way [CES86]. This technique

is applied to models rather than working systems, so its usage is limited to the parts of the system that can be accurately modelled. For a CPS it is not always possible to prove some requirements algorithmically, because the physical part of the system cannot be accurately predicted, and even the software cannot be analysed [Ram00] unless it has been automatically generated from a model by a verified tool. Statistical Model Checking is a technique based on the randomized sampling of simulations of a stochastic model. This variant of Model Checking is more suitable for CPSs because it accounts for their randomness [CZ11].

- **Testing** is a common verification process for software which aims to find bugs or defects in the systems [Ber07]. This is accomplished by executing some components of the system and checking that the behaviour of the execution is correct. Although testing a component can usually not be done exhaustively and therefore the lack of detected failures does not ensure the correctness of the system in general, it can expose deviations from the requirements [Ber07]. There are different testing levels depending of what is being tested: Unit testing for individual components, integration testing for component interactions, or application testing for the whole system. The tests can be manually created, or automatically generated based on black box techniques such as boundary value analysis or white box metrics such as code coverage [Lim09]. Analogously to software, hardware can also be tested, either as an abstract definition such as a Hardware Description Language (HDL) or a model, or with physical hardware. Model-Based Design (MBD) is a widely used method for developing CPSs [SWYS11, JCL11], and Model-Based Testing (MBT) allows testing the functionality of these models, which describe the software, hardware and the physical environment of the system [UPL06, JCL11].

- **Runtime Verification** is a technique which consists in analysing execution traces of the System Under Test (SUT) to check that a given specification is not violated.

2.2.1 Execution Levels

During the development and verification of CPSs, it is common to use X-in-the-loop simulations to bridge the gap between the system specification and the implementation [SH09]. There are four typical configurations defined for this:

- **Model-in-the-Loop (MiL)**: This configuration consists on testing the controller model along with the simulated plant model. The model executor usually uses high-precision floating point arithmetic in order to compute the most accurate results. Once the MiL test results are satisfactory, they can be used as a reference for comparison with subsequent execution levels. At the model level, the failures detected during an execution should be trivial to diagnose and fix for an engineer who is familiar with the tool and the problem.

- **Software-in-the-Loop (SiL)**: In this level, the executable code of the SUT is used, which can be autogenerated from the model used in the previous stage. The executable code is generated for the development or testing machine rather than the target processor. The software is still tested with a simulated plant and hardware. The goal is to detect computation errors such as overflow or division-by-zero, which may not have been detected due to the different arithmetic in the model executor.

- **Processor-in-the-Loop (PiL)**: Next, the executable code for the target processor, i.e. the code that will actually be used in the final product, is generated. While the software is still being interfaced with a simulated plant and hardware, the software itself runs in the target processor, possibly in an evaluation board or a virtual machine. Although the tests at this level are still not Real-Time (RT), they can reveal arithmetic errors or otherwise inadequate target-specific code that were not revealed when compiling for the testing machine.

- **Hardware-in-the-Loop (HiL)**: Finally, the product is verified with the software running in the target processor and a hardware setup which can emulate the real execution environment. The execution of the whole system will be done in RT, so under these circumstances the performance requirements and timing constraints of the system will be validated. Since execution must happen in real-time, this process might take a very long time (hours, days) to complete.

Level	Software	Arithmetic	Hardware / Plant	Time
MiL	Model	Floating point	Simulated	Non RT
SiL	Simulating processor	Fixed point	Simulated	Non RT
PiL	Target processor	Fixed point	Simulated	Non RT
HiL	Target processor	Fixed point	Physical / Emulated	RT

Table 2.1: Comparative of the X-in-the-Loop Levels

Figure 2.1 summarizes the X-in-the-loop phases and the artefacts involved on each stage. Table 2.1 compares the characteristics of each of the stages.

Although the X-in-the-loop paradigm is typically used in Model Based Development [SSSG14], the SiL, PiL and HiL simulation levels can still be implemented regardless of the software development methodology used, as long as hardware and plant simulators are available.

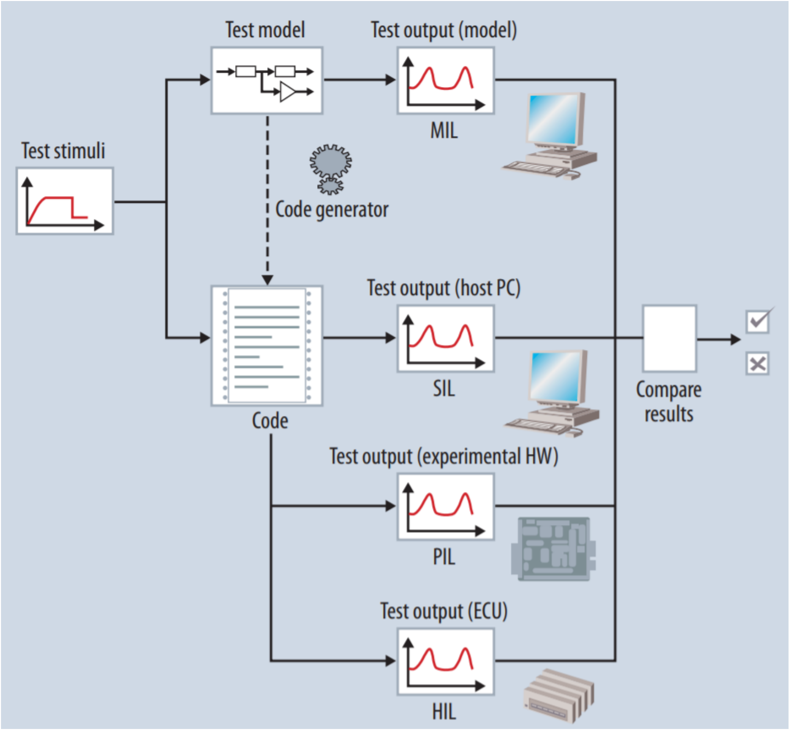


Figure 2.1: Summary of X-in-the-loop phases [SH09]

2.3 Non-Functional Verification

Besides verifying the functional aspects of a system, many domains require the verification of non-functional aspects.

Performance testing is the most explored form of non-functional testing, with the most commonly verified performance metrics for software being: Execution time, memory usage and energy consumption [ATF09, SDPMBS20]. Another relevant non-functional aspect are the Quality of Service (QoS) attributes of the system [ATF09, HCL19], such as response time, availability, reliability, etc. Other non-functional aspects that have been explored in the context of verification are: Security, usability and safety [ATF09].

In the context of CPSs, many of these aspects are particularly important. For instance, all of the previously mentioned performance metrics are highly relevant in embedded software, where computation resources and memory are usually scarce, and the energy consumption and execution times are often constrained (e.g. smart sensors and real-time systems respectively). On the other hand, QoS and security testing have also been explored in this domain, mostly in the context of cloud-based and Internet of Things (IoT) systems (e.g. [PTP+19]).

2.4 Metamorphic Testing

A common problem with CPS verification is the difficulty of obtaining exact specifications for these complex systems [KFK14], i.e., the oracle problem [BHM+14]. For CPSs, the derived specifications are often approximations that can lead to false positives or negatives in some cases (e.g. [KFK14]). For some systems, such as machine learning based ones, it may even be infeasible to define any sort of specification, since they are often applied to problems where the expected functionality of the system is not defined explicitly [XHM+11, DAS+18].

MT is a technique which adopts an alternative approach to alleviate the oracle problem: instead of verifying the correctness of an individual execution, MT exploits known input and output relations that should hold among multiple executions, which are known as Metamorphic Relation (MR) [CCY98].

A typical example of MT is the verification of a program that calculates the sine function by using the following known property: $sin(x) = sin(\pi - x)$. In this example, MT would allow us to test this program not by computing the sine separately for verification, but instead by generating any input pair x and $\pi - x$, executing the program with both inputs, and verifying that the outputs are equal. The properties that define the relations between the inputs and the outputs of the program are known as MRs. The common way of performing MT consists in generating a source test case (x) by using existing test generation techniques such as random testing, and then generating the corresponding follow-up test case based on the input relation ($\pi - x$). A possible application of this technique for a self-driving car would be verifying that the system behaves identically for multiple test cases traversing the same routes under different weather conditions (e.g. by applying rain or snow filters to the image, such as in [TPJR18]).

MT has been used in many domains, such as machine learning applications, web services, computer graphics, and compilers [SFSRC16]. This technique has also been successfully applied in the domain of CPSs, such as for testing Wireless Sensor networks [CCC+07], autonomous drones [LPMS17], or self-driving cars [TPJR18, ZZZ+18, ZS19].

2.5 SBSE

Search-Based Software Engineering (SBSE) is an approach in which Search-Based Optimization (SBO) algorithms are used to address software engineering problems [HMZ12]. Miller and Spooner [MS76] where the first to apply such optimization techniques to a software engineering problem [HMZ12], specifically in the software testing domain. The term SBSE was later coined by Harman and Jones [HJ01]. Since then, this approach has become increasingly popular in many software engineering problems, including those related to testing and verification. Such problems include test case generation [FA11], selection [AWM$^+$19], or prioritization [AWSE19], and even test oracle generation [TJTP20].

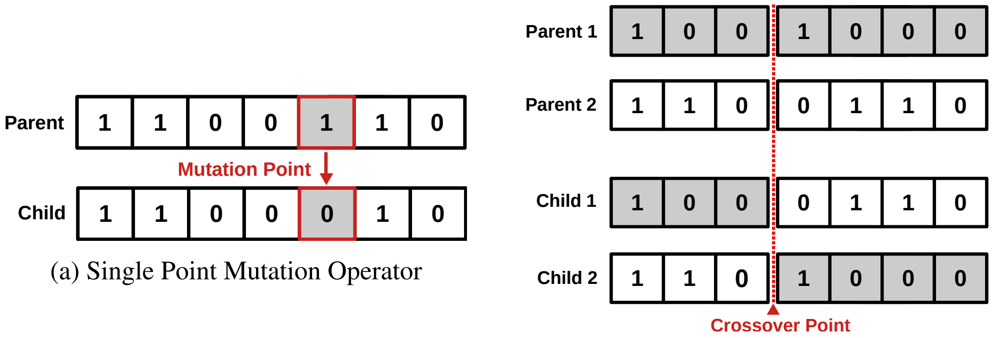
Harman and Jones identified the three key ingredients for applying SBSE to a problem [HJ01]: (1) The choice of *representation* for the problem, (2) the definition of *fitness functions* to guide the search, and (3) the *manipulation operators* to explore the search-space.

2.5.1 Algorithms

Random Search (RS) is the most basic search algorithm, which simply consists of randomly exploring the search-space for candidate solutions [HMSY10]. With this algorithm, the fitness functions are not used for guidance, and the only manipulation operator consists of producing new random candidate solutions from scratch. This technique will struggle to locate optimal solutions for problems where these are located at specific subsets of the search-space, since this algorithm will not focus on these areas.

Local search refers to the algorithms which address a single candidate solution at a time, "moving" from the current candidate solution to a neighbouring one in order to find a solution with the highest possible fitness [HMSY10]. These algorithms can efficiently traverse the search-space for increasingly better solutions with the guidance from fitness functions, until a *locally optimal* one is found. Examples of local search algorithms include Hill Climbing [SG06] and Simulated Annealing [KGJV83].

Global search algorithms, on the other hand, address many points of the search-space at the same time [HMSY10]. This approach is appropriate for problems with several local optima, since a local search might get stuck in a locally optimal solution and fail to locate the global best solution in the search-space. *Genetic Algorithms (GAs)* [Hol75, Gol89], which evolve a *population* of candidate solutions emulating Darwinian evolution, are the most well known form of global search algorithm [HMSY10].



(a) Single Point Mutation Operator

(b) Single Point Crossover Operator

Figure 2.2: Example Manipulation Operators

Genetic Algorithms

In Genetic Algorithms (GAs), the candidate solutions from the population, known as *individuals*, undergo selection, mutation, and reproduction processes, which guide the search towards a population with fitter individuals. The selection process decides which individuals get to live and/or reproduce to pass their genes to the next generation, and is usually guided by the fitness functions. One of the most commonly used selection operators is Tournament Selection, where the individuals are randomly introduced into “tournaments”, and the fittest ones are chosen as parents for reproduction [MMGG95]. In the reproduction step, crossover operators are used to combine the genes of the parents selected from the previous generation. Mutation operators can also be used in order to introduce changes to the individuals from the next generation and enable a broader exploration of the search-space.

As an example, consider a test case selection problem, where a test suite comprises a set of test cases which can be selected or not: $TS = \{t_1, t_2, ..., t_N\}$. An adequate solution representation could be a bitset $S = \{s_1, s_2, ..., s_N\}$, where the value of the bit s_n would indicate whether the test case t_n is selected or not (e.g. 1 if it is selected, 0 if it is not). For this solution representation, we could use the single point mutation operator depicted in Figure 2.2a, which randomly selects and flips one of the bits from the individual. On the other hand, in order to combine the genes of the selected for reproduction, parents, Figure 2.2b illustrates a single point crossover operator. This operator randomly chooses an index from the solutions, known as the crossover point, and recombines the genes from both parents using that point as a boundary. The remaining tasks to adapt this problem for SBSE would be defining an appropriate fitness function (e.g. maximize the effectiveness versus cost ratio of the test suite) and choosing a search algorithm.

2.5.2 Multi-Objective Search

For some problems where SBSE may be applied, there might be *multiple* different optimization objectives, often conflicting with each other to some degree [Deb14]. For the previous test selection example, it is common to define the problem with at least two objectives [CGT$^+$20]: (1) minimize the cost of the test suite, and (2) maximize the effectiveness (e.g. Fault Detection Capability (FDC)) of the test suite.

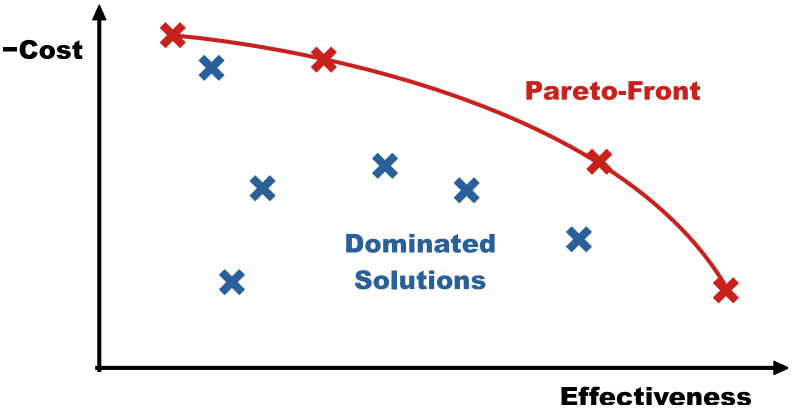


Figure 2.3: Test Selection Pareto-Front

Figure 2.3 shows an example set of solutions for a test selection problem with two competing objectives, plotted in 2-D space depending on their fitness for the two objectives. Note that the cost is negated in order to convert it to a maximization objective, so the best solutions are the ones further away from the axes' origin. Even though there are multiple conflicting objectives, and their relative weight might not be clear, there are still some solutions that are clearly better than others. A solution is considered *dominated* when there is a different solution with an better fitness for at least one objective and equal or better fitness for the rest. In these cases, the solution that is dominated would be uninteresting, since the dominating solution has a superior fitness regardless of how we weigh each objective. Non-dominated solutions are called *pareto-optimal*, and the set of these solutions form the *pareto-front*.

Many multi-objective search algorithms are pareto-based, i.e., they return the set of solutions from the *pareto-front* they find. Commonly used multi-objective search algorithms include the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [DPAM02], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [ZLT01] and Pareto Envelope-based Selection Algorithm II (PESA-II) [CJKO01]. These algorithms have been widely adopted for solving test case generation, selection and prioritization problems [YH07, AWM$^+$17a, AWM$^+$19]. A common drawback of many of these algorithms is the lack of scalability for a large number of objectives

[PKT17]. Other algorithms such as the Non-dominated Sorting Genetic Algorithm III (NSGA-III) [DJ13] and Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [ZL07] have been developed to address this issue. Panichella et al. developed an algorithm tailored to the test case generation problem called DynaMOSA (Many-Objective Sorting Algorithm with Dynamic target selection) [PKT17]. This technique has been adopted as the default algorithm by Evosuite, a popular Java test suite generation tool [FA11, FA12].

2.5.3 Co-Evolution

Co-evolution consists of having two or more populations evolving concurrently, with fitness functions that make the populations affect each other. This concept is inspired by natural phenomena where species influence each other's evolution, such as predators and prey or hosts and parasites [AHH04]. This technique can be used to generate a competitive process between opposing entities (e.g. mutants and test suites), which makes the training set for each of them dynamic, and thus prevents getting stuck at local maxima [Hil90].

The first application of this technique for SBSE was an approach to co-evolve mutants and test cases by Adamopoulos et al., where the mutants act as prey and test cases act as predators [AHH04]. Arcuri and Yao proposed an approach to generate programs from their specifications by co-evolving the program itself and its unit tests [AY07, AY14]. The same authors also developed an automatic software bug fixing approach where, given the program specifications and a buggy program, the test cases and programs would co-evolve as predators and prey respectively, eventually leading to a fixed program [AY08]. More recently, Terragni et al. developed a co-evolutionary technique for test oracle generation, where one population prioritizing false positive reduction and another prioritizing false negative reduction share genetic material with each other periodically [TJTP20].

Chapter 3

State of the Art

This chapter presents an overview of the state of the art for the topics related to this thesis, highlighting the most relevant publications. Furthermore, Section 3.4 performs a critical analysis of of this state of the art and points out various research opportunities.

3.1 Automation of CPS Development

Due to the long development life-cycles of Cyber-Physical Systems (CPSs), the adoption of DevOps techniques in this domain is being studied in order to speed up the related processes [ABG$^+$20]. The DevOps approach is an evolution of the agile methodology, which advocates short and incremental release iterations with intermediate reviews over a longer and more complete release cycle [LRK$^+$19]. This approach can increase the frequency and quality of new software releases [LRK$^+$19], but the overhead of many current development practices, as well as regulation and security concerns, make it infeasible for CPSs in practice [ABG$^+$20]. In this regard, the adoption of automated tools for the development process, which is a domain known as Continuous Software Engineering (CSE), is paramount to implement the DevOps approach in the context of CPSs [OEC17]. In order to develop such automation tools, different design techniques have been explored in the literature, such as model-based engineering [CW19], Digital Twins [UQES20], or microservices architectures [OEC17].

3.1.1 Model-Based Engineering

Model-Based Design (MBD) is a popular CPS development paradigm in which abstract models are used to describe the functionality of the system [SWYS11, JCL11]. In order to perform design and development tasks with this paradigm, many techniques such as Model-Based Testing (MBT) have been proposed [UPL06].

Combemale and Wimmer discussed a full model-based DevOps approach encompassing all the related tasks, including testing, integrating, deploying, monitoring, execution and monitoring [CW19]. Their approach includes the use of runtime models in order to link the system's operational data with the models and support model-based techniques at the operation stage [CW19].

Most of the research work, however, is focused on specific DevOps tasks. For deployment, Ferry et al. proposed a deployment and orchestration approach for smart Internet of Things (IoT) systems which uses the GeneSIS modelling language, a novel model-based representation which encompasses design and run-time activities [FNS$^+$19]. Petrovic and Tosic defined a model-based approach for the deployment and adaptation of container-based fog applications [PT20]. As for monitoring, Monisha et al. presented a secure and robust framework for monitoring healthcare systems based on CPS modelling techniques [MRB19]. Regarding verification and validation, Balasubramaniyan et al. proposed a tool-supported approach for CPSs which combines multi-objective evolutionary optimization and model checking techniques [BSB$^+$16].

3.1.2 Digital Twins

Digital Twins are virtual real-time replicas of a system built with a combination of functional models and monitored data, i.e., they stimulate a real CPS in its current state [GS12]. This technique allows an accurate digital replication of the system, enabling processes such as monitoring, verification, or failure prediction and adaptation [GS12, GV17].

Ugarte et al. presented an approach which uses Digital Twins as an enabler for DevOps in the context of Cyber-Physical Production Systems (CPPSs) [UQES20]. More specifically, the Digital Twins can be used to collect data and predict system malfunctions, as well as to verify newly implemented functionality in a controlled (digital) environment [UQES20].

3.1.3 Microservices Architectures

One if the common solutions for CSE is the microservices architecture, which consists in having a set of small and self-contained tools, each of which providing a specific functionality [OEC17]. This architecture allows the seamless integration of new functionalities or changes to a system with minimal complexity and error-proneness [OEC17, GL18]. This design is particularly useful for CPS tooling because the DevOps systems would have to adapt to the heterogeneous nature of this domain. Indeed, different CPS vendors will use different development environments, communication technologies, monitoring systems, etc., so custom tooling will have to be seamlessly integrated into each environment [ABG+20].

Many practitioners have already experimented with microservices architectures for developing the CPS themselves. Butzin et al. proposed adopting microservices architectures for IoT systems due to the ability to easily integrate services from different providers [BGT16]. Later, Lu et al. proposed the use of a microservices architecture following specific Application Programming Interface (API) design patterns to enhance the security of IoT systems [LHWM17]. Alam et al. proposed a Docker-based architecture for highly modular time sensitive systems, which aims to simplify their management while ensuring reliability and recoverability [ARF+18].

On the other hand, microservice architectures have also been proposed for specific DevOps tasks. Kang et al. proposed a container-based microservice-style DevOps approach for cloud infrastructure deployment tasks [KLT16]. Miglierina and Tamburri presented Omnia, a factory pattern for automatically composing monitoring systems intended to reduce the entry-cost of adopting monitoring solutions for CSE [MT17]. Waseem et al. recently performed a systematic literature review on microservice architectures in DevOps, and identified 47 primary studies and 50 related tools [WLS20]. They identify research addressing quality attributes, design, development and deployment, testing, and monitoring of microservice architectures for DevOps [WLS20].

3.2 Oracle problem

The oracle problem manifests itself when defining a test oracle for a particular system becomes infeasible. This happens when there is no accurate specification for the behaviour of the system, which can be the case in several domains [BHM+14]. This section describes the typical instances of the oracle problem and the techniques that can be used to alleviate it.

According to [BHM+14], test oracles can be classified into three main categories:

- **Specified oracles**. These are oracles based on specifications of the System Under Test (SUT), including partial system behaviour specifications such as assertions or models.

- **Implicit oracles**. This type of oracle relies on common knowledge which can be used to identify incorrect behaviour, such as buffer overflows or uncaught exceptions being errors (i.e. blatant faults).

- **Derived oracles**. These oracles distinguish the system's correct and incorrect behaviour based on information derived from artifacts such as documentation, system execution traces or other versions of the system.

The following subsections describe the most notable techniques from each of these categories.

3.2.1 Specified Oracles

These oracles are based on partial or complete specifications of the SUT. Any form of specification is implicitly an oracle which can be used to judge whether the system behaves according to it [BHM$^+$14].

The main challenges for this type of oracles is usually either the lack of a formal specification or its imprecision. Indeed, many specification formalisms rely on abstraction, and comparing real system behaviour and outputs against an abstract specification is not always straightforward. Indeed, this is often the case for CPSs, where the specifications are often imprecise or incomplete [KFK14].

The following are some of the most common specification formalisms used across various domains.

Model-Based Specifications

Model-Based specification languages model the system as a collection of states and operations to alter them, with preconditions and postconditions which constrain the behaviour.

The Abstract State Machine (ASM) specification language generalizes Finite State Machines (FSMs) by allowing states to be arbitrary structures (mathematical structures, which may contain data, functions, events, etc.) [Bör10]. Such a model can be used to check whether a given system output is possible, and a mismatch with the model can be identified as an invalid behaviour.

Many other model-based specification languages, such as VDM [FL09] or Z notation [SA92], can also express invariants, which can be directly derived into test oracles. Li and Offutt evaluated different test oracle strategies for model-based testing with varying precisions (more precise means checking more variables) and frequencies (more frequency means checking variables more frequently) [LO16]. They concluded that higher precision or frequency does not necessarily result in more effectiveness [LO16].

Other oracles simply check that the system's transitions and outputs of an execution comply with the idealized state machine. In this case, a failing verdict is emitted when the state machine does not accept the real system's execution. For CPSs, such modelling can suffer from spurious divergences due to the non-determinism from physical processes. Gay et al. presented a model steering approach which consists in backtracking the model state and performing small corrective changes to match the behaviour of the SUT in order to increase the tolerance of these oracles for CPSs [GRH16].

Zhang et al. presented a framework for modelling the *uncertainty* of CPSs with the Unified Modeling Language (UML) [ZAY$^+$19b]. This framework, named *UncerTum*, enables the development of CPS models using the UML Uncertainty Profile (UUP) and extensions to the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). Their modelling of uncertainty is based on the U-Model [ZSA$^+$16], which comprises: (1) *Belief Model*, which describes the subjective worldview (Beliefs) of BeliefAgents; (2) *Uncertainty Model*, which describes the uncertainties themselves (type, locality, effect, risk, lifetime, etc.); and (3) *Measure Model*, which describes how the uncertainty is measured (in terms of probability, fuzziness, ambiguity, etc.). Their toolchain supports standard UML-based Model-Based Testing (MBT) practices using the UML Testing Profile V.2.

Algebraic Specifications

Algebraic specification languages describe first-order logic axioms which the system should follow. They are typically used to describe Abstract Data Types (ADTs), which contain both data and operations over that data, and are commonly used for specifications of classes and interfaces in object-oriented designs [PZ14].

Temporal logic is a family of languages used for the formal specification of system behaviours [BFFR18]. The most common and basic form of temporal logic is Linear Temporal Logic (LTL), which supports propositional formulas and qualitative (e.g. always, never, eventually, until) temporal operators [Pnu77].

There are variants of LTL which consider a *quantitative notion of time*, i.e. they consider the timestamps of the events rather than only taking the ordering of the events into account. This is important for CPSs, and particularly for those with real-time requirements. Metric temporal logic (MTL) [Koy90] is one of these variants, in which temporal operators are annotated with intervals.

Signal Temporal Logic (STL) is a variant which is particularly relevant for CPSs, because instead of events it considers collections of *signals* in the continuous time and value domains as traces [MN04]. This can be seen as the combination of the metric temporal logic extension of LTL and the addition of continuous numerical value predicates. There is also an extension which adds first-order logic operators to

STL called Signal First-order Logic (SFO) [BFHN18]. This formalism is suitable for verifying hardware components and CPS models in a simulation environment (e.g. Simulink models).

It must be noted that the more expressive variants of LTL usually result in an increased oracle cost, i.e. more resources (time, memory...) are needed to verify the properties. In fact, some researchers deliberately restrict the expressiveness of their specification language in order to be able to generate more efficient monitors. Menghi et al. proposed a method to generate online test oracles for Simulink models based on properties expressed in Restricted Signals First-Order Logic (RFOL), a language which can handle continuous signals while being feasible to implement in practice [MNGB19].

Boufaied et al. presented SB-TemPsy, a model-driven approach for checking signal-based temporal properties for CPSs [BMB+20]. Their solution includes a novel Domain-Specific Language (DSL) to express the most common requirement types for CPSs, as well as a trace-checking procedure which uses model-driven techniques to reduce the cost of checking complex system properties.

More recently, Boufaied et al. developed a taxonomy of the various types of signal-based properties used to specify the behaviour of CPSs, describing the expressiveness and other features of each formalism [BJB+21].

Program Assertions

An assertion is a check placed at a certain point of the program which ensures that a particular boolean expression is true. These checks can also be considered test oracles [BHM+14].

Compared with other specification languages, assertions reduce mismatches between the specification and the implementation, since they are specified directly in the implementation (source code) and can therefore directly define relations over the program variables [BHM+14].

The *design by contract* approach extends regular assertions by defining pre and post conditions that check contracts between a *client* and a *supplier*. For instance, Java Modelling Language (JML) is a specification language for annotating Java classes and interfaces with behavioural information (e.g. requisites for the input, constraints for the returned values, etc.) [LBR06]. This specification language was also extended by Murphy et al. to allow the definition and checking of metamorphic properties at the method level [MSK09b].

3.2.2 Implicit Oracles

Although not fully universal, implicit oracles exploit common knowledge in the context of the SUT (domain, programming language, etc.) in order to derive test oracles.

In order to find implicit anomalies, a common approach is to employ fuzzing, i.e., generating and feeding random inputs to the program under test [MFS90], since the implicit oracles usually represent rules which must hold for any given input.

The first works to employ implicit oracles consisted in systems to detect common concurrency errors, such as deadlocks, livelocks, or data races [Kel76].

Another instance of such oracles is related to performance testing with metrics such as the CPU time used by an application [MHH13], since degradations of such metrics can reveal system errors.

Another common technique is implemented by profilers which can automatically detect memory leaks of a program by tracking references and reporting objects which have become unreachable without having been freed [ABS94].

Newer studies have also derived oracles for Machine Learning (ML) systems by exploiting common knowledge from their domain to estimate their confidence. Stocco et al. exploited implicit domain-knowledge to predict misbehaviours from autonomous vehicles with image-based Deep Neural Network (DNN) driving systems [SWCT20]. Their approach, SelfOracle, employs an autoencoding technique in order to estimate the confidence level of the DNN [SWCT20]. This technique exploits the knowledge that DNNs are more likely to misbehave on unseen inputs. A newer research exploits the attention maps from the driving system in a similar way, using the assumption that uncommon attention maps correlate with a lower confidence from the DNN [SNdT22].

3.2.3 Derived Oracles

When specified test oracles are unavailable, new specifications can be extracted by using additional information such as system execution traces, other versions of the system, documentation, etc.

Pseudo-Oracles

Pseudo-oracles are alternate versions of the SUT developed independently from the original, which can be used as test oracles in order to reveal faults in *non-testable* programs [DW81]. According to Weyuker [Wey82], *non-testable* programs are either: (1) Written to determine the given answer, thus the correct answer is not known and cannot be verified, or (2) programs which produce so much output that it is impractical to fully verify it.

This approach is similar to the N-version programming technique for fault-tolerant computing, where different implementations of the software are executed in parallel and a voting mechanism decides the final output should the outputs differ [Avi95].

In [Fel98], genetic algorithms were used to produce diverse versions of the same software that can be used for this purpose. This technique was adopted by Sprenkle et al. for evaluating the HTML output of a web application by comparing it with the previous version [SPE$^+$07].

Regression Testing

Regression testing aims to detect whether modifications made to a new version of the SUT have interfered with any existing functionalities. With this technique, a previous version of the SUT is assumed to be correct and used as an oracle for the previously existing functions.

A system update can belong to one of the following categories, an this affects how regression testing is performed:

1. **Corrective changes**. These are modifications which are intended to only fix errors without adding or removing functionality. In this case, all the program can be tested for regressions, checking that no disruptions are made to the existing functionality, and also that the fix covers all the cases which can trigger the bug [GBHS10].

2. **Perfective changes**. These updates add new functionality which was not present in the previous version. For these changes, regression testing can be applied to the older functions of the SUT, which ought to remain unchanged, and the oracle and test suite should then be augmented to handle the new functionality [HA08].

3. **Specification changes**. This class of modifications is often not differentiated from perfective changes. This is because when such a change happens, the functionality with a changed specification can be considered as a new function, while the versions compliant with an older specification can just be considered deprecated functionalities.

Mezzetti et al. applied regression testing for Node.js (JavaScript) libraries in order to detect breaking changes to their public interface [MMT18]. Their approach learns models of the library interface in order to detect typing changes, which are a common source of breaking changes between minor (perfective) and patch (corrective) updates that are often detected late due to JavaScript's dynamic nature [MMT18]. However, the most common approach is using automatic regression test suite generation tools, such as the Java tool Evosuite [FA11]. Such tools generally try maximize the code coverage and similar metrics while maintaining a small test suite size, providing a high cost-effectiveness for regression testing after corrective changes [FA12]. Such test suites would need to be expanded or refactored in the case of changes to the system functionality.

Oracles Based on Textual Documentation

Textual documentation can be written as natural language descriptions of the system functionality or with a structured format, such as an API documentation. Either way, these documents describe the intended functionality of the system, and thus, they can potentially be used as test oracles.

It has been acknowledged that documentation is one of the key factors to improve the quality of the software development process, including the testing phases [Par11]. Nevertheless, using documentation as a basis for test oracles is usually limited to humans (i.e., manual testing) due to the difficulty of interpreting natural language specifications, and testing based on extracted documentation (e.g. auto-generated from the source code) is circular because it assumes that the code from which the documentation was extracted was correct in the first place [Par11].

Parnas et al. proposed the use of fully-formal tabular expression-based specification formats from which test oracles can be automatically derived [PP98, FPTO11]. Another approach for automated test oracle extraction from documentation written in a (restricted) natural language was proposed by Schwitter [Sch02]. This language was PENG, a subset of the English language with controlled vocabulary and grammatical structures which can be deterministically transformed into first-order predicate logic [Sch02]. More recently, Motwani and Brun [MB19] presented an approach to generate test oracles and executable tests from structured natural language specifications, and applied their approach to generate tests which verify compliance with the JavaScript specification. The generated test suite was able to identify faults in two different JavaScript implementations.

Blasi et al. proposed MeMo [BGE+21], an approach to automatically identify Metamorphic Relations (MRs) from JavaDoc comments. They employ a two-step approach for this: First, they identify sentences which describe MRs in the JavaDoc. Then, the identified MRs are translated from the natural language specification to an executable MR.

Metamorphic Testing

Oracles for MRs are an alternative that can be used when the expected output of a system cannot be known. This is often the case, for example, when verifying Machine Learning applications, due to factors such as a large number of inputs, unpredictability of the system, and inexactitude of the outputs (e.g. an image classifier may not classify all the inputs correctly) [DAS$^+$18]. Rather than checking the correctness of the program output, metamorphic oracles check whether multiple executions of the program under test fulfil certain conditions referred to as metamorphic relations. [SFSRC16].

MRs can consist of any type of relation between executions. For example, $sin(x) = sin(\pi - x)$ is a metamorphic relation for the *sin* function. These relations can be expressed in general as rules of the form ***if*** *<relation among inputs/outputs>* ***then*** *<relation among inputs/outputs>*, where the first part corresponds with the *source test case* and the second part corresponds with the *follow-up test case* [SDTC17]. For the previous example, we could write: ***if*** $sin(x) = y$ ***then*** $sin(\pi - x) = y$.

These kinds of oracles have been applied in many fields, such as Web applications [JJB$^+$19], embedded systems, machine learning applications [DAS$^+$18, ZZZ$^+$18, ZS19], numerical programs and code generators [LAS14, BBSB20, DELT17], among others [SFSRC16]. As for examples of the industrial adoption of metamorphic testing, this technique has been successfully applied to the Data Collection JavaScript Library of the Adobe Analytics software in order to find bugs related with specific versions of browsers or their JavaScript engines [WTZC18]. In [LGÁW15], a model-based metamorphic testing approach was used in NASA's Data Access Toolkit, which is an interface to query a large database of telemetry data, in order to verify that its API returns the correct data for the input queries.

In [STDRC17, STDRC18], Segura et al. proposed the use of metamorphic testing for identifying performance-related bugs in programs rather than just functional failures. Such MRs had previously been used by Chan et al. [CCC$^+$07] to find non-functional faults based on the energy consumption of Wireless Sensor Networks. Johnston et al. applied performance MRs based on the statistical distribution of web page load times in order to identify bugs in a tag manager library [JJB$^+$19]. Ahlgren et al. implemented an approach for tackling the test oracle and test flakiness problems in Facebook's simulation-based testing platform by employing statistical MRs [ABB$^+$21]. Donaldson et al. defined semantics-preserving source code transformation patterns to test OpenGL graphics shader compilers [DL16, DELT17]. In order to locate faults on code generators, Boussaa et al. employed a statistical approach with MRs based on the performance and resource usage of the generated code [BBSB20].

Several works have applied metamorphic testing for the verification of CPSs. Chan et al. defined MRs to verify both functional and non-functional properties of Wireless Sensor networks [CCC$^+$07]. Lindvall [LPMS17] et al. presented a model-based approach for testing autonomous drones, in which they derive MRs that check the drone's state (e.g. disarmed, takeoff, flying) and its transitions. More recently, several techniques based on metamorphic testing have emerged for testing autonomous driving systems [TPJR18, ZZZ$^+$18, ZS19]. Several of them focus on equivalence MRs, such as simulating environmental conditions like rain or fog [TPJR18, ZS19]. More recently, Deng et al. presented a rule-based metamorphic testing framework for testing autonomous driving systems, which enables the conversion of human-written rules to MRs using natural language processing techniques [DZZ$^+$22]. This framework also encompasses automatic metamorphic test generation using image transformations.

Metamorphic Test Cases

As for the generation of metamorphic test cases, the most common approaches for obtaining source test cases are random test generation and using an existing test suite [SFSRC16]. Empirical studies have also concluded that randomly generated data may be more effective than manually designed test suites, although a combination of both can be even more effective [CKLT04, SHBRC11]. In [GB03], a technique for automatically generating test data for MRs called *Automatic Metamorphic Testing (AMT)* is presented. This technique consists in analysing the source code of the program, written in a subset of C, and using *Constraint Logic Programming* to find test data which violates a given MR. Other test generation approaches consist in maximizing the branch coverage of the program by using genetic algorithms [BS11, CCL$^+$12] or symbolic execution [DGZ13].

Discovery of Metamorphic Relations

One of the main downsides of this approach is the necessity of defining metamorphic relations. Some generic metamorphic relations can be easily defined, but research has shown that defining good metamorphic relations requires knowledge of the problem domain [SFSRC16]. Nevertheless, it is possible to discover metamorphic relations by using various automated approaches.

Many publications have suggested reusing abstract patterns in the input or the output relations in order to derive concrete MRs for a particular system [SPTRC18, ZSCT18]. Chen et al. presented an approach and tool to achieve the semi-automation of this process called METRIC [CPX16]. This tool generates candidate MRs based on category-choices over the input domain, which the user can accept or reject [CPX16].

Sun et al. later extended this approach with METRIC+, which considers both the input and output domains when generating candidate MRs [SFP+19].

Another approach is to exploit Metamorphic Relation Patterns (MRPs) and define reusable templates, which may be instantiated to generate MRs for different systems. Segura et al. defined such patterns for query-based systems [SDTRC19]. Their work assumes that the system performs standard operations such as filtering, ordering and pagination, and defines patterns such as: "Shuffling the search terms should result in equivalent results" [SDTRC19]. This is a generalization of their previous work, where they defined such patterns specifically for REST APIs [SPTRC18]. Xie et al. developed METTLE, a Metamorphic Testing (MT) approach for testing unsupervised ML systems which relies on 11 generic MR templates [XZC+20]. They propose patterns such as shuffling the dataset or scaling the coordinate system [XZC+20].

Liu et al. proposed an approach to construct new metamorphic relations by combining several existing ones, with the idea that the composite MRs would embed all properties of the original MRs into a single, more effective one [LLC12]. Arrieta recently explored the use of using composite MRs for testing image recognition deep learning systems, but concluded that composite MRs where less effective than their components [Arr22b].

Some fully-automated MR discovery approaches have also emerged lately. Blasi et al. presented an approach for identifying equivalence MRs from natural language documentation [BGE+21]. Kanewala et al. proposed static analysis techniques to predict MRs based on the the control flow graph of the program under test using graph kernels [KB13, KBBH16], and Duque et al. recently reproduced their approach with positive results [DTPRK22]. On the other hand, Zhang et al. presented a MR prediction approach based on the method's control flow graph and an improved radial basis function (RBF) neural network [ZZPL17].

Another popular approach is the use of search-based methods over program executions. Goffi et al. proposed an approach for inferring equivalent method sequences, which can be expressed as equivalence MRs [GGM+14]. Zhang et al. presented Metamorphic Relation Inferrer (MRI), a tool to discover MRs which can be expressed as equalities of polynomial expressions [ZCH+14]. Their approach employs Particle Swarm Optimization (PSO) and a set of program executions in order to infer MRs. Zhang et al. later developed AutoMR, another approach based on PSO with a greater expressiveness and effectiveness than MRI [ZZC+19]. AutoMR is capable of discovering both equality and inequality MRs, and includes additional filtering steps to remove invalid or redundant MRs, resulting in a smaller pool of more effective MRs [ZZC+19].

Oracles Derived with Machine Learning

Another possible approach consists in having the oracles infer the rules of the system behaviour based on the execution traces by using *Machine Learning* techniques.

Neural Networks

Shahamiri et al. employed Artificial Neural Networks (ANNs) to learn the behaviour of the program and generate oracles which detect anomalies [SKIH11, SWKIH12].

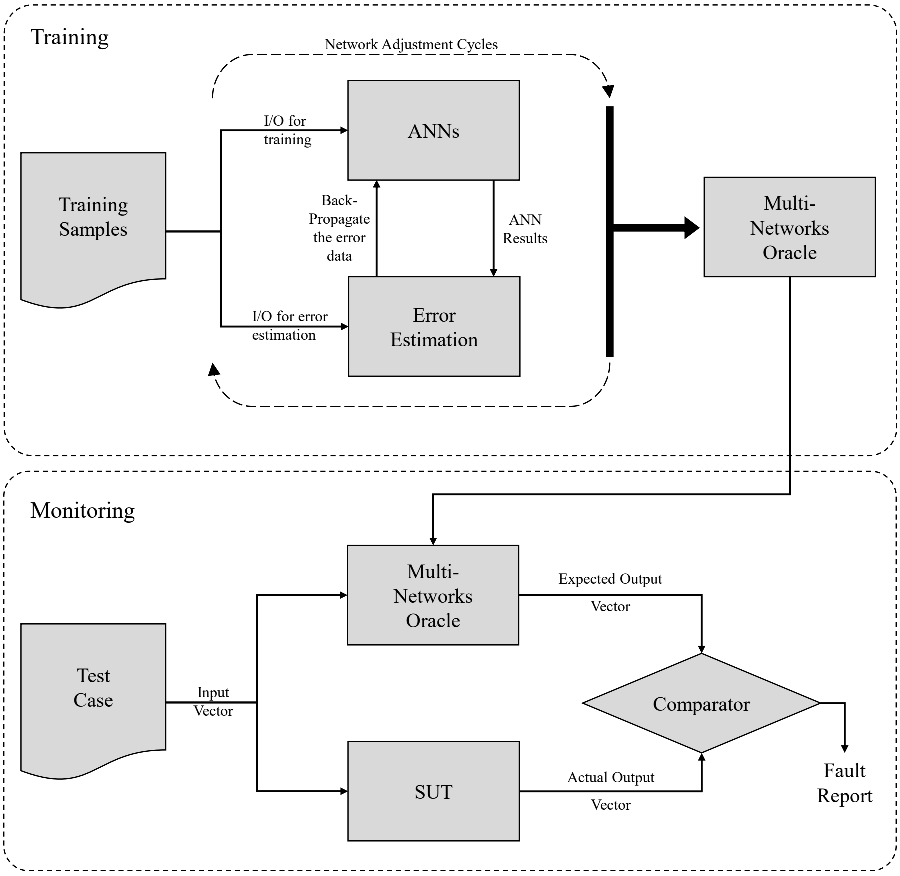


Figure 3.1: ANN Oracle Generation Process [SKIH11]

Figure 3.1 shows an overview of the process for generating an ANN-based oracle (the initial training data generation is omitted). This approach is divided into three main steps:

1. **Generate training data.** The first step is to generate training data for the oracle. The datasets are generated separately for each output, considering only the inputs

2. **Train the ANNs.** Then, the ANNs are trained with the generated data. Rather than using a single network for modelling the behaviour of the whole system, this approach uses an independent ANN for each output of the system. The I/O relationships of the system are analysed before the initial training data generation, so each ANN only considers the inputs that affect their corresponding output.

3. **Apply the trained ANNs.** Finally, the trained ANNs can be integrated with the system as an oracle. The verdict from the ANN oracle is calculated as the Mean Squared Error (MSE) between the monitored system outputs and the ANN outputs, which means that the oracle can calculate a quantitative verdict.

More recently, Dinella et al. presented TOGA, a neural approach to generate both assertion and exception-checking oracles for Java programs [DRML22]. Unlike the previous approach by Shahamiri et al., this technique uses the *test prefix* and the *method context* to generate the oracles. Here, the *test prefix* is the code which sets the system to the state where the method under test is called, whereas the *method context* encompasses surrounding method signatures and documentation. The oracle generation process is broken into two steps:

1. **Exceptional Oracle Classifier.** First, a neural model performs the binary decision task of choosing between expecting an exception or not. In the case of expecting an exception, the exception-expecting oracle can generated immediately, which has the form of a *try...catch* block surrounding the method under test. Otherwise, TOGA proceeds to use the Assertion Oracle Ranker to produce a regular assertion oracle.

2. **Assertion Oracle Ranker.** This second neural model chooses a single element between a set of candidate assertions. The candidate assertions are constructed over combinations of the method return value, global variables, and a dictionary of the most common constant values.

Support Vector Machine

On the other hand, Chen et al. presented a different approach which uses Support Vector Machines (SVMs) and mutants for learning property monitors [CPS18].

Their approach has the following steps:

1. Randomly execute the original system for n times and collect normal traces.

2. Construct mutants from the original system.

3. Execute mutants and collect mutant traces.
4. Identify abnormal traces by comparing mutant traces with the normal ones.
5. Collect a set of positive feature vectors from the normal traces.
6. Collect a set of negative feature vectors from the abnormal traces.
7. Learn a SVM classifier.
8. If stopping criteria is not met, restart with additional data.

After the SVM is generated, Statistical Model Checking (SMC) is used to estimate the probability of the classifier's correctness. If the probability is low, the process can be restarted with different randomly sampled data. Otherwise, the generated SVM can be used to classify the live monitored data and raise an alarm when the behaviour does not classify as normal.

Unsupervised Learning

Afzal et al. developed Mithra, an approach for detecting anomalies in CPSs based on telemetry data [ALGT21]. Their approach employs a multivariate time series clustering over system execution traces to discover the different correct behaviours of the CPSs. The output from this clustering is then used to derive robustness oracles which identify anomalous behaviours from the system.

Invariant Generation

Ernst et al. presented Daikon, a dynamic invariant detection approach which reports likely invariants that hold over all the observed program executions [EPG$^+$07]. This approach checks for a set of 75 different possible invariants, which can be instantiated over the program variables at method entry or exit (i.e., method pre and post conditions). Their implementation supports several programming languages or raw data inputs, as well as multiple output formats for the invariants.

Gupta and Rybalchenko developed InvGen, an invariant generator which relies on user-provided templates and constraint-based techniques [GR09]. Given a user-provided template of parameterized linear inequalities at a specific program point, InvGen computes parameters that instantiate the template into a concrete invariant that is valid. In order to do this, InvGen collects dynamic information for the minimized program using both concrete and symbolic execution, and converts it into a series of constraints. Then, a constraint solver computes parameter values which satisfy these constraints, and these parameters are used to instantiate the templates into concrete invariants.

Terragni et al. proposed GASSERT (Genetic ASSERTion improvement), an automated oracle improvement technique based on genetic algorithms which can generate Java program assertions [TJP+20]. This approach employs samples of correct and incorrect program states in order to evolve program assertions, with the goal of minimizing false positives and false negatives. A novel co-evolutionary approach they propose evolves two populations of assertions concurrently, one prioritizing false positives and the other false negatives. Their tree-based assertion representation allows arbitrary numeric or Boolean sub-expressions in the generated assertions. They also leverage the OASIS (Oracle ASsessment and Improvement) test oracle assessor [JCHT16, JCHT18], which is a tool that combines test case generation to reveal false positives and mutation testing to reveal false negatives in the oracle. The false positives and negatives identified by OASIS are incorporated into the set of correct and incorrect states, and a new iteration of the GASSERT evolutionary algorithm is run.

More recently, Molina et al. presented EVOSPEX, an approach for automatically generating postcondition assertions for Java methods [MPAF21]. Similar to GASSERT [TJTP20], this approach also relies on samples of correct and incorrect subject method executions and a genetic algorithm. However, their approach uses a different representation for postcondition assertions, in which they are modelled as variable-length conjunctions of assertions with a single relational (e.g. $==, ! =, >, ...$) or quantification (*all* or *some*) operator. Since this approach employs an assertion representation which involves quantification, it is especially suited for reference-based classes with strong representation invariants, such as container types like ArrayList or HashMap.

Molina et al. later developed SPECFUZZER, a dynamic invariant detection technique which employs grammar-based fuzzing [MdA22]. This approach generates invariants by following these steps: (1) The *Grammar Extractor* generates a specific grammar for the assertions of a given class, (2) the *Grammar Fuzzer* produces candidate assertions from the class grammar, (3) the *Invariant Detector* filters out assertions invalidated by a test suite using Daikon [EPG+07], and (4) the *Invariant Selector* uses a mutation analysis mechanism to cluster and rank assertions in order to discard weak or redundant ones.

3.3 Test Optimization

Test optimization refers to various techniques that may be used to reduce the cost and maximize the effectiveness of the testing process. This is particularly important in the domain of CPSs, where testing is very costly, taking over 50% of the development resources in some domains [BG11]. This section describes the relevant work on various test optimization techniques.

3.3.1 Test Selection

Test selection is an optimization technique which focuses on selecting a subset of the available test cases in order to maximize the cost-effectiveness of testing. Test minimization is a variation of test selection where redundant test cases are permanently eliminated from the test suite. In contrast, test selection usually does not imply that the tests which are not selected are redundant. Thus, test minimization can effectively be considered a special case of test selection [Har11, YH12].

Kazmi et al. presented a systematic literature review of 47 studies addressing regression test case selection [KJMG17]. In that work, it is revealed that the majority of relevant publications between 2007 and 2015 employ “mining and learning” approaches, with Genetic Algorithms (GAs) being the most popular techniques [KJMG17].

In practice, there are often multiple criteria to consider when selecting test cases, which is why test selection is often formulated as a multi-objective problem [YH07]. Yoo and Harman proposed the use of pareto-efficient multi-objective algorithms for this task, and presented an empirical evaluation demonstrating this approach [YH07]. Their experiments employed the Non-dominated Sorting Genetic Algorithm II (NSGA-II) algorithm and the statement coverage and computational cost objective functions [YH07]. They also proposed vNSGA-II, an island extension of NSGA-II which uses multiple sub-populations in order to achieve wider pareto-frontiers. The same authors later applied multi-objective search algorithms to tackle the test minimization problem [YH10]. They proposed Hybrid NSGA-II (HNSGA-II), a hybrid algorithm which complements with NSGA-II with a greedy algorithm in order to achieve a higher efficiency [YH10].

Panichella et al. presented DIversity based Genetic Algorithm (DIV-GA), a multi-objective genetic algorithm which injects new orthogonal individuals during the search process in order to encourage diversity [PODPDL14]. In their experiments, they found that DIV-GA outperforms both greedy and a multi-objective algorithm (vNSGA-II [YH07]) which employ only coverage and cost objectives.

Pradham et al. proposed an approach for test case selection when there is a limited time budget for executing test cases [PWAY16]. They evaluated various multi-objective and weight-based search algorithms, and concluded that the Strength Pareto Evolutionary Algorithm 2 (SPEA2) outperformed every other technique in both real-world and large-scale artificial case studies [PWAY16].

In the domain of CPSs, Arrieta et al. proposed a black-box test case selection approach using NSGA-II and six different objective functions based on anti-patterns and similarity measures [AWM$^+$19]. The specific objective functions they explored were: Instability, discontinuity, growth to infinity, output minimum and maximum difference, input-based test similarity, and output-based test similarity [AWM$^+$19]. Their study concluded that these black-box metrics outperform white-box approaches for test selection in this domain [AWM$^+$19]. A newer study which used the same approach and case studies proposed effective seeding strategies to help the search converge [AVAS22].

Beyond GA approaches, some research has explored fuzzy expert systems [XGKS14] and PSO [DSPBA13]. Furthermore MBT techniques have also been applied for test case selection. In this regard, Hemmati et al. explored similarity-based test case selection approaches for test suites generated from state machines, and evaluated them on two industrial case studies from the embedded systems domain [HAB13].

3.3.2 Test Prioritization

Test prioritization is an optimization technique loosely related to test selection. In contrast to test selection or minimization, test prioritization aims to order a set of test cases to be executed [KIJT18]. The goal is to execute highly significant test cases first in order to achieve some objective, such as revealing faults, earlier [KIJT18]. The main advantage over test selection is the ability to dynamically incorporate (and insert in the correct order) new test cases while the test suite is being executed [KIJT18].

Khatibsyarbini et al. recently performed a systematic literature review on test case prioritization approaches in regression testing [KIJT18]. Their study revealed that search-based techniques are the most popular, which include GAs [BS19, BLC13, BKH$^+$15] and greedy algorithms [LHH07, AWSE16b], among others. The most common objectives for such approaches are coverage [BLC13, BKH$^+$15], fault [RUCH99, YL12] or requirement [SHD16] targets [KIJT18]. Some approaches also exploit historical data from previous test execution results [KP02, LCTK13, CKL16]. Henard et al. performed a comparison of black-box (e.g. input similarity based) and white-box (e.g. coverage based) approaches, and concluded that the results were comparable in terms of achieved Failure Detection Ratio (FDR), and that there was also a great overlap in the specific faults located [HPH$^+$16]

Coverage-based prioritization approaches aim to achieve a high coverage of the system under test as early as possible [YH12]. This involves analysing the SUT, and using measures such as branch or statement coverage [HZZ$^+$15]. Yoo et al. identified two main types of coverage approaches: total and additional [YH12]. Total coverage approaches simply prioritize the test cases based on their individual coverages, while additional approaches aim to achieve a complete coverage earlier by prioritizing test cases with a higher coverage on yet uncovered parts of the SUT [YH12]. Khatibsyarbini et al. observed that coverage-based approaches tend to employ greedy algorithms [KIJT18].

While most approaches target regular software rather than CPSs, some recent studies have tackled this domain. Arrieta et al. presented a test case prioritization approach for configurable CPSs using weight-based search algorithms [AWSE16b]. Their approach converts a multi-objective problem into a single objective one by assigning relative weights to the normalized test execution cost and Fault Detection Capability (FDC) objectives [AWSE16b]. They later extended this approach to support CPS product variants [AWSE19]. This extension also considers multi-level testing for CPSs, since many non-functional requirements (e.g. timing constraints) are only testable at the Hardware-in-the-Loop (HiL) level [AWSE19]. On the other hand, Mossige et al. proposed a test case scheduling approach to enable running tests concurrently on multiple machines [MGS$^+$17]. Their solution employs constraint programming in order to provide the schedule which will result in the fastest possible execution of the test suite while ensuring consistent access to shared resources (e.g. hardware devices) [MGS$^+$17]. Zhang et al. presented an approach for generating and minimizing CPS test suites based on uncertainty [ZAY19a]. Their solution includes various multi-objective test generation and minimization strategies which aim to reduce the number of test cases while maximizing various uncertainty-based coverage measures [ZAY19a].

3.3.3 Metamorphic Testing

Since MT employs multiple test cases at once, generic test optimization techniques may not be straightforward to apply, although many concepts can be carried over.

One of the most straightforward approaches to optimize MT is adopting existing test case generation or selection approaches, and applying them to the source test cases. Most contributions on MT use randomly generated test cases or pre-existing test suites for the creation of source test cases, but some other approaches have been explored [SFSRC16].

Batra and Sengupta proposed a source test case selection approach which employs GAs to maximize the code coverage [BS11]. Chen et al. proposed another approach based on GAs, which aims to diversify the test inputs by partitioning the input domain and trying to cover as many of these partitions as possible [CCL$^+$12]. Sun et al. proposed an approach for generating source test cases based on their associated path constraints, which are obtained through symbolic execution [SLF+22]. The path distance among test cases can also leveraged to guide the prioritization of source test cases [SLF+22].

On the other hand, Arrieta recently proposed an approach for selecting metamorphic test cases for deep learning systems based on the cost and the confidence of the test inputs, considering both the source and the follow-up test cases [Arr22a].

Another approach for optimizing MT is selecting the most effective MRs and discarding ineffective or redundant ones.

Many studies have analysed correlation between the features of MRs and their effectiveness, and pointed out specific characteristics which might indicate whether they are strong or weak. For instance, studies have shown that MRs with more dissimilar executions in their corresponding source and follow-up test cases (in terms of white-box coverage) tend to be stronger [CHTZ04, CZC13]. On the other hand, Chen et al. concluded that black-box approaches are inadequate for identifying effective MRs [CHTZ04]. However, Mayer et al. later pointed out that the black-box criteria employed by Chen et al. were inadequate, and suggested other criteria [MG06]. For instance, they identified that MRs consisting of simple equalities tend to be weak, as they are susceptible to many types of false negatives (e.g. if the result is multiplied by a factor, or if a constant is added to the result) [MG06].

As for automated approaches, Spieker and Gotlieb proposed Adaptive Metamorphic Testing (ART), an approach which uses a reinforcement learning technique (contextual bandits) to select MRs which are more likely to reveal faults [SG20]. On the other hand, Srinivasan et al. proposed the use of coverage and fault based criteria to prioritize MRs [SK22].

Composite MR generation approaches can also be considered a type of MT optimization. In their work on this topic, Liu et al. explicitly present achieving the same effectiveness with fewer MRs as a potential advantage of composite MRs [LLC12].

3.4 Analysis of the State of the Art

This section aims to identify the potential research avenues for CPS development process automation, focusing on test automation and alleviating the oracle problem.

The adoption of Continuous Software Engineering (CSE) tools is a necessary step to bring DevOps methods to the CPS domain. Some partial solutions for related tasks, such as deployment [RVT+18, FNS+19, PT20], monitoring [VCHB+18, MRB19], or validation [BSB+16, MLM18] have already been proposed, but a holistic and generic architecture for DevOps which addresses all the concerns specific to CPSs is yet to be defined. Combemale and Wimmer described a more complete DevOps architecture, but it is tailored for model-based development, and is therefore not suitable for CPSs which do not follow a model-based design extensively [CW19]. A unified architecture should address the concerns specific to CPSs, such as heterogeneous tooling or high reliability requirements. A major challenge in this regard is the need to integrate components from different vendors and tailor the DevOps system to the requirements from different industry practitioners (existing development models, tool-chains, etc.) [ABG+20]. Microservice architectures can provide a flexible solution for this by defining the DevOps system as a set of loosely-coupled and composable tools [OEC17, GL18].

Regarding CPS testing, many systems require the verification of performance or Quality of Service (QoS) metrics, for which defining a test oracle is particularly challenging due to their unpredictability and non-determinism. When it comes to performance testing, most approaches so far rely on either setting threshold values which can never be violated, detecting known types of problems, or comparing the results against existing data [JH15]. In some cases, it is possible to execute the same performance tests multiple times in order to detect inconsistencies among executions, or even performing comparisons within the results of the same execution [JH15], but the most common approach is to perform regression testing against an existing baseline (e.g., a previous version of the system) [FJA+15, JH15, SHNF15]. Metamorphic testing can be an alternative to regression oracles which can mitigate the oracle problem by defining output relations among different test case executions. The application of performance metamorphic testing has not been explored yet in many domains [STDRC17, STDRC18], and most of the existing examples of this approach address testing software applications rather than CPSs [SFSRC16]. It would be interesting to define MRs over performance or QoS metrics in order to detect failures in CPSs, which is an avenue that has not yet been explored extensively by any previous work. Another potential contribution could be comparing MT with other alternative test oracles, such as approaches based on Machine Learning (ML).

One of the major challenges of MT is the definition of *correct* and *effective* MRs, which should have as few false positives and negatives as possible. The definition of such MRs is known to require practical experience with the SUT and its application domain [SFSRC16], which results in a high cost in the form of time spent by domain-experts to define the MRs. Some semi-automated approaches which aim to assist the test engineers in this task have been proposed [CPX16, SFP+19], but they can only mitigate this issue to a degree, since they require domain-specific information and user input. Template-based approaches have also been evaluated with great success in the domains of query-based systems [SDTRC19] and machine-learning [XZC$^+$20], but such templates are domain specific, and significant work needs to be done in order to develop such templates for a new domain. Some fully-automated MR generation approaches have also been presented. AutoMR uses an approach based on PSO which can discover polynomial numeric MRs of a function, and also includes processes to filter out incorrect and redundant MRs [ZZC$^+$19]. Despite this approach being fully-automated and generic, it is limited to numeric, and specifically polynomial properties only. Furthermore, while the generated MRs can be expected to be *correct*, they are not necessarily *effective*, since the search process does not use incorrect input/output pairs, and thus does not have a concept of false negatives. A fully automated approach which is more generic and considers false negatives in order to identify more effective MRs could help the adoption of automatic MR discovery tools. GAssert is a tool which implements such an approach, using Genetic Programming (GP) in order to generate Java program assertions based on correct and incorrect test executions [TJP$^+$20], but this technique would have to be adapted to the domain of CPSs and to MT in particular.

Regarding the optimization of MT, several approaches have been proposed for source test case generation or selection [BS11, CCL$^+$12, SLF$^+$22], as well as for MR selection [SG20, SK22]. However, these approaches only consider a single dimension for the optimization: Source test case selection approaches do not consider the MRs to be used nor the follow-ups generated, whereas MR selection approaches do not consider the source and follow-up test cases. Arrieta recently proposed an approach which considers both source and follow-up test cases [Arr22a], but it is tailored for deep learning systems and does not consider the selected MRs (e.g. maximizing MR diversity). A more general approach for selecting metamorphic tests considering all factors (sources, follow-ups, and MRs) could target more complex objectives at once (e.g. maximizing MR diversity and input diversity for both source and follow-up test cases), which may allow for a more effective test selection process for MT.

Chapter 4

Theoretical Framework

This chapter presents the objectives (Section 4.1) and hypotheses (Section 4.2) for the research. Then, a high-level overview of the theoretical framework for metamorphic testing and its intended application context is shown, and the areas of contribution are highlighted (Section 4.3). Finally, a description of the case studies employed to evaluate the contributions is provided (Section 4.4).

4.1 Research Objectives

The goal of this thesis is to **develop and evaluate Metamorphic Testing (MT) based methods to alleviate the oracle problem for Cyber-Physical Systems (CPSs)**. Based on the research opportunities identified after analysing the state of the art, we defined the following objectives:

- **Objective 1:** Develop and evaluate a MT approach to alleviate the oracle problem for CPSs were the effectiveness of specified oracles is limited.
- **Objective 2:** Identify and document useful Metamorphic Relation Patterns (MRPs) that could aid the development of Metamorphic Relations (MRs) for CPSs from different domains.
- **Objective 3:** Implement and evaluate a tool for automatically generating MRs using Genetic Programming (GP).
- **Objective 4:** Implement a metamorphic test selection tool and evaluate different metaheuristics and fitness functions in order to maximize the resulting cost-effectiveness of MT.

4.2 Research Hypotheses

The research work will try to prove the following hypotheses:

- **Hypothesis 1:** MT can detect faulty behaviours of CPSs that cannot be detected with specified oracles. This corresponds to research objective 1.
- **Hypothesis 2:** The definition of MRPs can help define effective MRs for different domains. This corresponds to research objective 2.
- **Hypothesis 3:** The use of GP can allow the automatic generation of effective MRs. This corresponds to research objective 3.
- **Hypothesis 4:** Multi-objective search algorithms can be used to increase the cost-effectiveness of MT by selecting a subset of the metamorphic tests. This corresponds to research objective 4.

4.3 Overview of the Theoretical Framework

Figure 4.1 depicts the theoretical framework for metamorphic testing, which is applicable for the verification of CPSs. Section 4.3.1 describes the application context in which this framework is intended to be applied, whereas Section 4.3.2 describes the framework itself.

4.3.1 Application Context

CPSs are inherently complex systems which include both software and hardware components [Lee08, Alu15]. During the lifecycle of these systems, the software is constantly evolving to adapt to new requirements, fix vulnerabilities, or correct bugs [AGA+20]. However, with the existing engineering practices for CPSs, releasing and deploying new software versions to production is a time-consuming and error-prone activity.

The high cost of developing new software for CPSs comes from the need of manual intervention for many key processes which could potentially be automated, such as configuration, deployment, monitoring and verification. In other domains, such as web development, DevOps practices have streamlined these processes and minimized the time to market for software changes. Unfortunately, these practices are yet to be adopted in the domain of CPSs.

The adoption of DevOps techniques for CPSs is a largely unexplored subject, so there is no standard to base on for defining the related methods and tools. On this note, we first developed a taxonomy in order to elicit the requirements for a CPS DevOps pipeline, and we applied it in two industrial case studies [AGA+20]. Furthermore, we later proposed a microservices-based design for this pipeline [GAA+21b] which

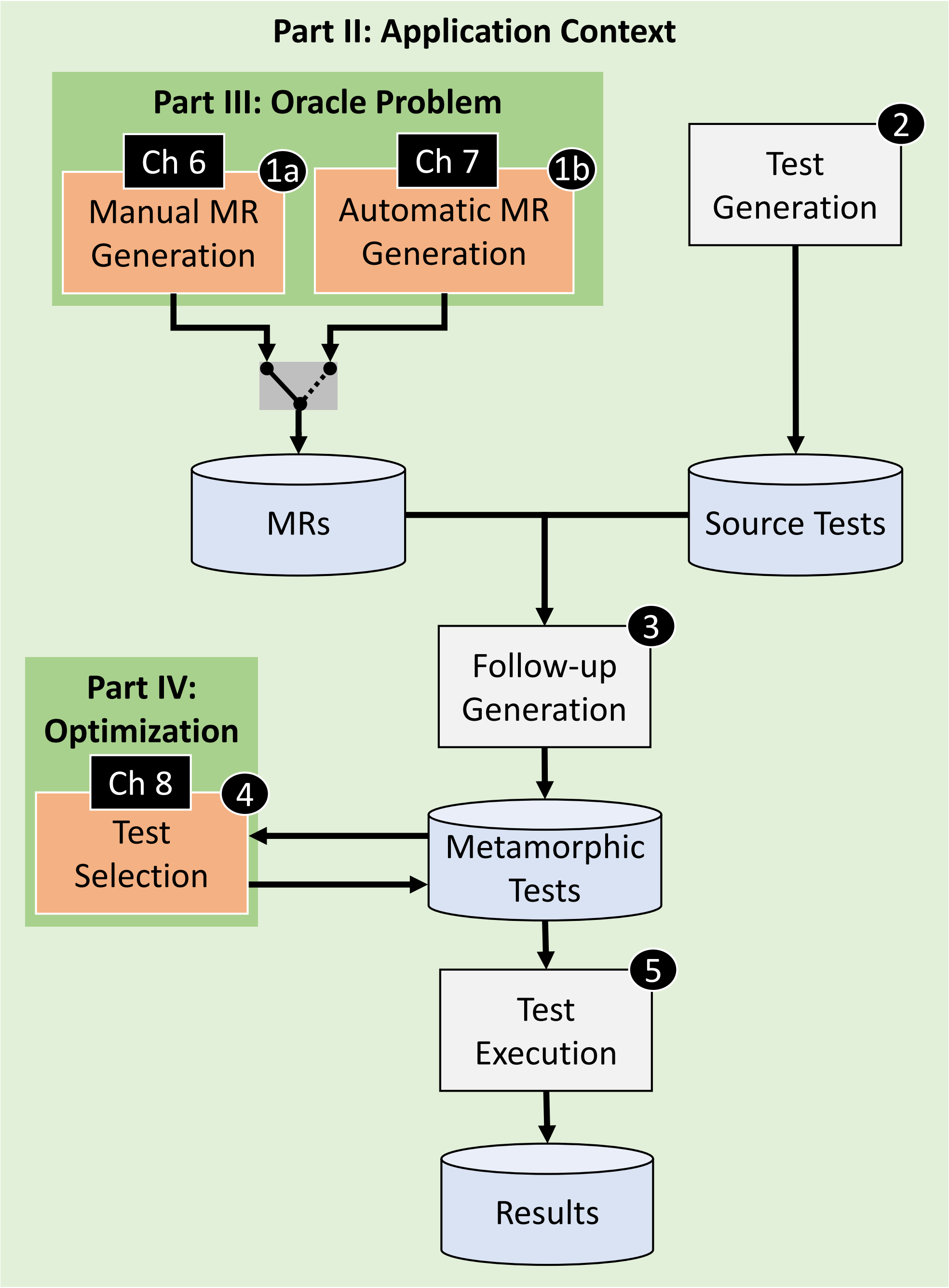


Figure 4.1: Overview of the theoretical framework and the contribution points of this dissertation

addresses these requirements. Part II of this thesis describes the proposed DevOps framework for CPSs in detail, and the specific problems that need to be solved.

In this thesis, we focus on the verification sub-problem of the DevOps framework, and more specifically in the test automation problem. One of the main issues for automating testing is the *test oracle problem* [BHM$^+$14], which refers to the inability to determine whether a test outcome is correct or incorrect. This problem forces the reliance on human oracles in many CPSs, resulting in manual or semi-manual testing processes. We evaluate MT as a solution to this problem, and we identify and address some of the gaps in the state of the art for this technique.

4.3.2 Metamorphic Testing Framework

In this section we describe the framework for MT, which is based on the existing literature on this technique [CCY98, SFSRC16, STZC20]. The MT process comprises 5 main steps (numbered rectangular boxes), and the specific contribution points are highlighted in green and orange, corresponding with the Parts and Chapters were the contributions are presented in this paper. The research objectives specifically target steps 1a (Objective 2), 1b (Objective 3), and 4 (Objective 4), while Objective 1 aims to develop and evaluate the overall framework in the context of CPSs.

The steps to perform metamorphic testing for CPSs include the following:

- **Step 1: MR Generation.** The first step is the generation of MRs to use as automated test oracles. This task usually requires advanced domain expertise in order to get correct and effective MRs, and it is usually not obvious whether an MR will be effective at detecting failures or not before evaluating it. Because this is a complicated and highly unexplored process, we evaluate two alternative approaches. This step outputs the MRs for the Systems Under Test (SUTs).

  - **Step 1a: Manual MR Generation.** The most common way of generating new MRs for a system is to define them manually. Besides having practical experience with the SUT and its domain [SFSRC16], this often requires some imagination, since MRs are often not mentioned in requirement documents. MRPs can help this process by allowing the reuse of common ideas for MRs.

  - **Step 1b: Automatic MR Generation.** An alternative approach to generate MRs is to employ automated techniques. These can be based on static [KB13, KBBH16] or dynamic [ZCH$^+$14, ZZC$^+$19] analysis of the system, as well as other artifacts like system documentation [BGE$^+$21]. On the one hand, these approaches can potentially solve the reliance on human imagination and expertise. On the other hand, these approaches can be difficult to implement and costly to apply, and automatically generated oracles may be prone to false positives [JCHT16, TJTP20].

- **Step 2: Test Generation.** The second step is the generation of test cases. The typical workflow for MT is to generate source test cases first, and then apply some transformations to generate follow-up test cases for each MR such that the input relation is satisfied. With this formulation, test cases can be generated using traditional test generation strategies, including random testing, black-box techniques (e.g. input diversity-based) [ND12, AWM+17b, AWM+17a], and white-box techniques (e.g. coverage-based) [ND12, MNBB16, MNBB18]. This step remained out of the scope of this thesis. The output of this step are source test cases for the SUTs.

- **Step 3: Follow-up Generation.** For each MR and source test case combination, it is then necessary to generate one or more follow-up test cases such that the output relation of the MR can be checked. This is often a straightforward task, provided that a rule for transforming test cases such that the input relation is satisfied can be easily derived. This step remained out of the scope of this thesis. The inputs for this step are the MRs (input relations) and source test cases, and the outputs are follow-up test cases. We call each source and follow-up test case pair for a specific MR a *Metamorphic Test*.

- **Step 4: Test Selection.** As an optional optimization step, a subset of the generated Metamorphic Tests can be selected in order to minimize the cost of execution while ideally maintaining most of the failure detection capability. This selection is typically done based on the features of the test cases [AWM+19]. The inputs for this step are the available Metamorphic Tests, and its output is a subset of them.

- **Step 5: Test Execution.** Finally, the Metamorphic Tests are executed in the SUTs, and the results are collected. For CPSs, the test executions can happen at many levels, such as Software-in-the-Loop (SiL), Hardware-in-the-Loop (HiL), or even the real operation environment [SH09, AWSE16a]. This step usually incurs the highest cost, and for environments such as HiL, manual intervention is often required. Some execution environments, particularly those that involve networking or physical components, may be inherently non-deterministic. In such cases, the same tests might have to be executed multiple times [GM07, GMSF07]. This step remained out of the scope of this thesis. The inputs for this step are the Metamorphic Tests and the MRs, and the outputs are the test results (pass or fail verdicts).

4.4 Case Studies

In order to evaluate the proposed testing approaches, we employ two different CPSs: An industrial case study from the elevation domain (Section 4.4.1), and an open-source case study from the automotive domain (Section 4.4.2).

4.4.1 Multi-elevator Installations

The first case study is an industrial multi-elevator system provided by Orona, one of the leading companies from the sector in Europe [Oro22]. This case study is used in Chapters 5, 6, 7, 8 and 9.

A multi-elevator installation is a complex CPS where the software and hardware components interact with the goal of transporting passengers safely while optimizing certain Quality of Service (QoS) metrics. These metrics to optimize vary depending on the customer needs, and include aspects such as the waiting times of the passengers or the energy consumption of the system.

Architecture

Figure 4.2 shows a simplified view of the system architecture. As the image shows, this is a distributed CPS, where most of the hardware components are interconnected via Controller Area Network (CAN) bus. These components include the traffic master, the individual elevator controllers, and various user interface elements, such as displays and elevator call buttons.

Among all these components, the traffic master is the computational unit in charge of managing the overall passenger flow of the installation. This component has many software modules, including the elevator dispatching algorithm, which decides which elevator should attend each call. The dispatching algorithm has a high impact on the QoS metrics of the installation. There are many different dispatching algorithms which optimize for different QoS metrics, but the one most commonly used at Orona is a deterministic rule-based algorithm called Conventional Group Control (CGC). The source code of this algorithm has been written in C in order to make it portable to different targets, including embedded systems and desktop hardware (for simulation-based testing).

Unlike other optimization systems, such as source code compilers, the dispatcher algorithm needs to make *predictions* of where the passengers will arrive, so the QoS metrics need to be optimized based on incomplete information. In general, the choices of the algorithm are based on the most common passenger behaviour patterns. Because of this, there is no objective definition of which dispatching behaviours or obtained

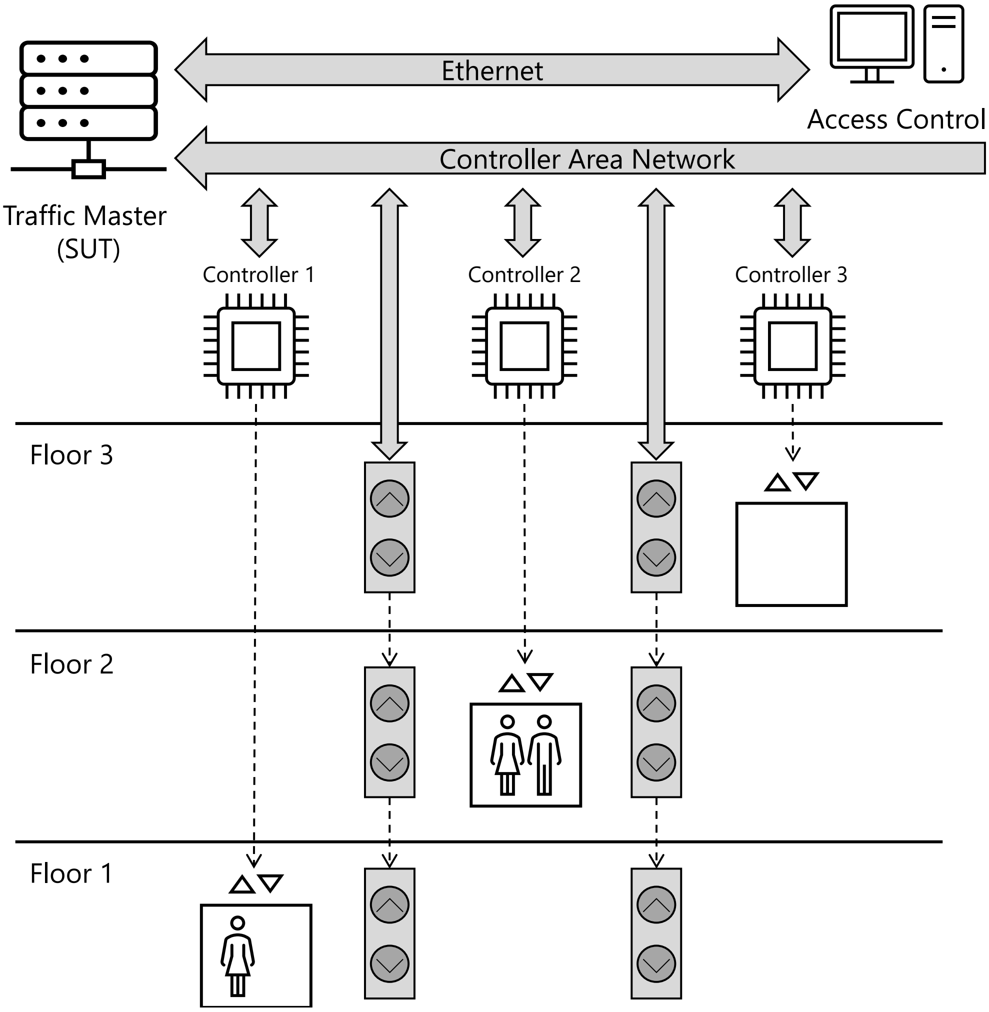


Figure 4.2: Architecture of Multi-elevator Installations

QoS measures are valid for a specific scenario, as tradeoffs need to be made when choosing which elevators to send for each passenger call. This is one of the reasons why there are not many automated test oracles available for this system.

Execution Environment

Figure 4.3 illustrates the current software development process at Orona. After the initial requirements elicitation step, the software is developed at the SiL level, using a simulator to mock the environment and hardware components of the system. After a release candidate has been developed and verified at the SiL level, the software is deployed into a HiL testbench, where most of the real hardware is used and the tests are run in real-time. Finally, the system is deployed into the real elevator installation,

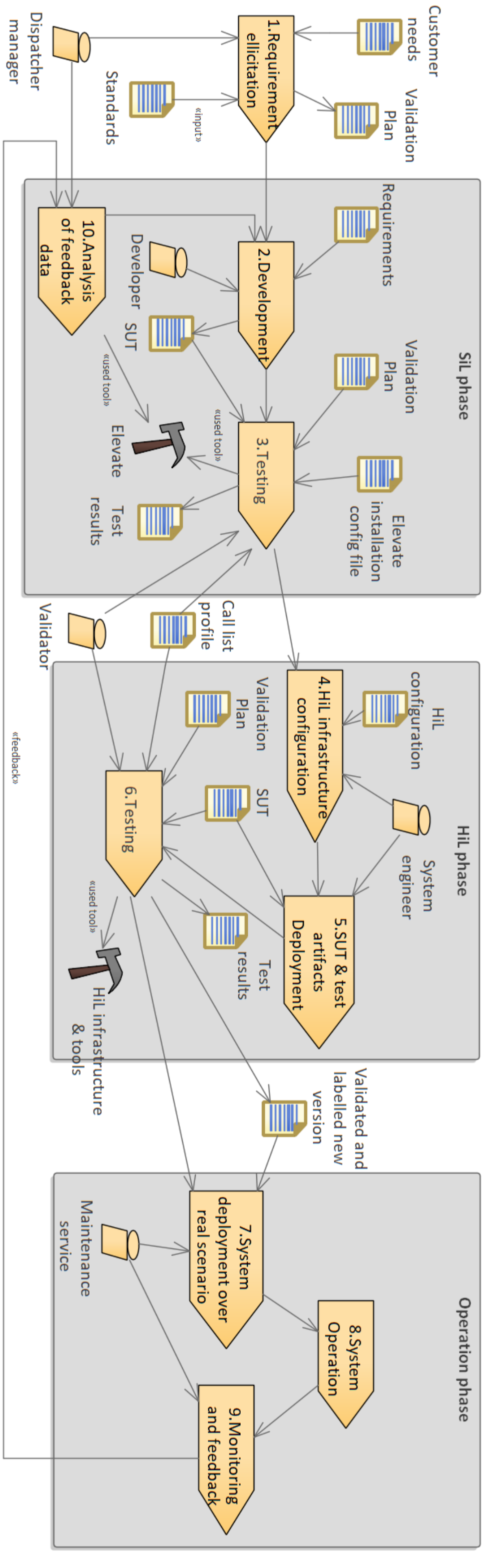


Figure 4.3: Software Development Process at Orona [AGA+20]

but is still monitored in order to detect issues and send feedback to the development team. The testing process, as well as fixing detected issues, becomes increasingly expensive at the latter phases of the process, so ideally most of the system's verification activities should be performed at the SiL level.

For the evaluation of different testing methods in this thesis, we employ the SiL level execution environment. The main tool employed in this environment is Elevate [Lim22], a domain-specific tool which can simulate entire multi-elevator installations, including the software, the hardware and the environment.

The test cases to be executed on Elevate are defined in two files. On the one hand, the *building data* is an XML file which describes the elevator installation, including the building layout and characteristics of the elevators. This includes information such as the number of floors, number of elevators and their initial positions, the maximum speed of each elevator, etc. On the other hand, the *passengers list* contains information about the passengers that will call the elevators to get to a different floor. For each passenger on the list, the file contains: (1) the arrival time, (2) the arrival floor, (3) the destination floor, and (4) the passenger's weight, as well as some extra information which determines the passenger's behaviour under various circumstances.

Testing Interface

To describe test cases for this system, we define the operation $serve(E, S, C)$, were E is a list of integers indicating the floors where the elevators are initially positioned, S is the nominal speed of the elevators, and C is a list of passenger calls.

The output of this operation is a QoS measure obtained by the elevation installation, which determines how effective the dispatching algorithm was for the scenario defined by the test case inputs. The goal of the test oracles is to determine whether the obtained QoS measures indicate a system failure or not.

Specifically, the QoS metrics employed for verifying the system are:

- **Average Waiting Time (AWT).** This is the average time the passengers have to wait from the moment they call for an elevator until an elevator opens its doors at the calling floor, measured in seconds. This is among the most important metrics to provide a good user experience [BAS15], and the main metric that the most common CGC dispatching algorithm aims to optimize.

- **Total Distance (TD).** The sum of distances traversed by all the elevators in the installation, measured in floors. This metric can reveal faulty behaviours such as sending multiple elevators for a single call or moving elevators for no reason, which would not necessarily have an impact on the AWT.

- **Total Movements (TM).** The total count of movements (engine start-ups) performed by all the elevators. This metric is also intended to detect failures similar to the ones described in TD.

4.4.2 Autonomous Driving System

The second case study is an open-source Autonomous Driving System (ADS) published by MathWorks [Mat]. This case study is used in Chapter 7.

The goal of this system is to autonomously drive a car from its initial position to a destination location through a path of reference waypoints. The quality of the ADS can be measured by QoS metrics such as the time to arrive to the destination or the degree of deviation from the reference path.

Architecture

Figure 4.4 depicts the overall architecture of this system. The subsystem on the top left (green) contains the reference waypoints that are passed to the ADS controller. On the bottom left (white) is the ADS controller which contains the software controlling the vehicle. The largest subsystem (blue) contains all the physical components of the system, including the internal vehicle dynamics and the environment interactions.

For any given initial and destination positions, the system will run a pathfinding algorithm do determine the shortest path between them. This pathfinding is based on a map of the road with reference road points which the vehicle is allowed to traverse. Once the path is determined, the autonomous vehicle must traverse the reference points in the path, moving in a straight line from its current position to the following reference point. The ADS will also receive a *nominal speed* as an input, which is the reference speed that should be maintained in stable conditions, although the controller may reduce it when the vehicle is adjusting its angle.

Besides the vehicle controlled by the ADS, known as the ego vehicle, other moving cars might be present in the road, which can force the ego vehicle to stop and wait until the path is clear in order to avoid collisions. In order to detect these other vehicles, the autonomous vehicle has a proximity sensor which the controller can use to detect obstacles.

Execution Environment

This system has been implemented entirely on Matlab/Simulink [DVDBSR13, Mat22], which is also the environment used to execute test cases on the system.

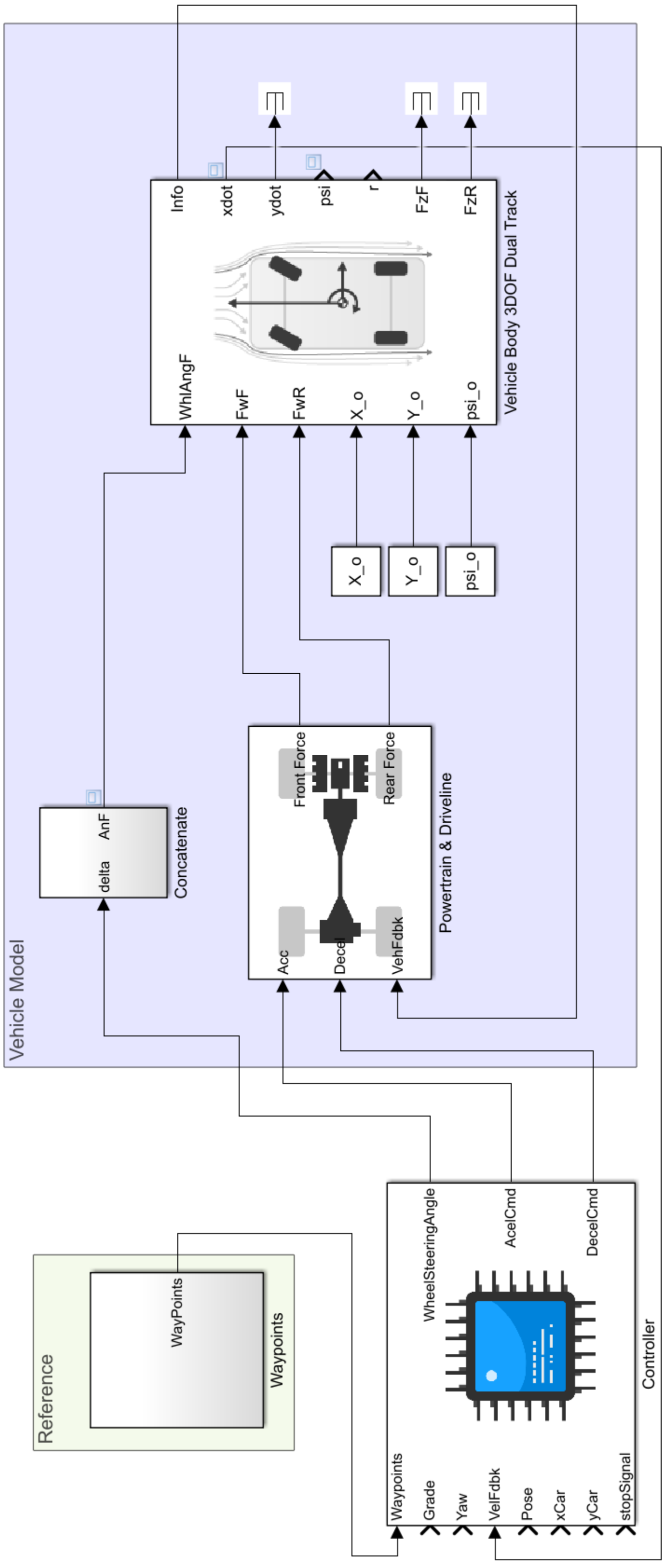


Figure 4.4: Architecture of the Autonomous Driving System

The scenarios for this system are defined by three main elements. Firstly, there is a map containing the reference points for the ADS, each of which is a joint between roads, as well as the ego vehicle's initial and destination points. Secondly, there are tunable parameters for the ADS, such as the nominal speed for the vehicle. Finally, the scenario may contain obstacles that the ADS must avoid, which in this case are other vehicles with an initial position and moving trajectory.

Testing Interface

We define the operation $move(P, p_A, p_B, S, O)$ in order to describe a test execution of this system. Here, P is a set of guidance points to follow, p_A is the initial position point, p_B is the destination point, S is the vehicle's nominal speed, and O is the set of obstacles in the environment (other vehicles).

The output of this operation is a QoS measure obtained by the ADS, which determines how effective it was for the scenario defined by the test case inputs. The goal of the test oracles is to determine whether the obtained QoS measures indicate a system failure or not.

The QoS metrics employed for verifying this system are:

- **Time To Destination (TTD).** The time it took from the start of the scenario execution until the vehicle is stopped at its destination point, measured in seconds. The ADS is expected to reach its destination as fast as possible, provided that the nominal speed is respected and there is no risk of collision against obstacles or deviating too much from the reference path.

- **Total Trajectory Offset (TTO).** This is the integral of the offset between the vehicle's angle and the reference angle. This metric determines how accurately the vehicle is facing the next reference point in its path. A high value of this metric may, for instance, indicate that the vehicle is having trouble to stabilize its direction because it is moving too fast. The ADS is expected to keep the value of this metric reasonably low in order to provide a safe and comfortable driving experience.

Part II

Application Context

Chapter 5

Requirements for CPS DevOps

Cyber-Physical Systems (CPSs) are holistic systems with typically long lifecycles, due to factors like the complex interactions between software and physical components and the strict safety requirements [Alu15]. Nevertheless, the software of these systems needs to be updated in order to keep up with customer demands, new regulations, or bug fixes, among other factors [AGA+20]. Currently, the development and deployment processes are not well integrated together, resulting in a high cost for deploying new versions of the CPS software.

This chapter aims to lay the groundwork for the adoption of DevOps practices in CPS software development by: (1) defining a taxonomy of the relevant concepts, and (2) using the taxonomy to elicit the requirements for a DevOps pipeline for CPSs. We base this work on two industrial case studies from the elevation and railway domains.

This work has been developed in a collaboration between Mondragon University, Ikerlan, Mälardalen University, Orona, and Bombardier Transportation [AGA+20]. The work for this thesis has been mainly focused on the validation subsystem of the DevOps taxonomy, and the facets of the taxonomy that do not involve a specific subsystem have been developed jointly.

5.1 Introduction

The design, development and deployment of CPS software is a complex and costly process which can take over a year from the start until it is deployed into the production environment [Alu15]. This is mainly due to the lack of synergy between the different phases of the process, as well as the lack of automation in many of the key steps.

In other domains, such as web development, DevOps [EGHS16] methods have already been adopted, and the entire process from development to deployment to production is completely streamlined. For CPSs, however, adopting DevOps methods requires a substantial change to ensure that the much stricter quality, dependability and safety requirements are met. Furthermore, some tasks such as the monitoring and replication of failures in operation can be much more challenging due to the high uncertainty of the dynamics between physical and software components.

To this end, we develop a taxonomy of the concepts relevant to capturing the requirements of DevOps systems for CPSs, in order to assist with the elicitation of such requirements for different CPSs. This work has been developed based on two industrial case studies from different domains, for which we also elicited DevOps requirements using the taxonomy. This chapter will focus mainly on the validation aspect of the DevOps system, which is the part that is relevant to this thesis.

The rest of this chapter is structured as follows: Section 5.2 describes the method used to define the taxonomy, Section 5.3 introduces the two case studies in which this work is based on, Section 5.4 presents the taxonomy itself, Section 5.5 discusses the related work, and Section 5.6 concludes the chapter and highlights future work in this line of research.

5.2 Taxonomy Definition Method

The taxonomy definition process was based on the guidelines provided by Ralph [Ral18]. The first step proposed by these guidelines is *choosing a strategy*. In this regard, "grounded theory and interpretive case studies" was chosen, although it was complemented with the personal experience from industrial and academic partners who had worked on the relevant CPSs for years.

Following this, we performed the *site selection* for the taxonomy. In this case, our sites are two different CPSs developing organizations and their corresponding systems: Orona (elevation domain) and Bombardier (railway domain). Both of these sites are relevant to our research questions, since they are large, industrial CPS developers with interest in adopting DevOps methodologies to their development process. Furthermore, the fact that the sites belong to different industry domains might indicate that the resulting taxonomy is generalizable to an extent.

Next, the *data collection* was performed on both sites. Two of different proposed data collection strategies were used [Ral18]. On the one hand, *direct observation* of the processes at each of the companies was employed, which involved reviewing their internal documentation such as process descriptions or source code comments. On the other hand, *interviews* to multiple domain practitioners with different positions and experience levels were employed to obtain additional information. The format of the interviews was semi-structured, since the development of a taxonomy on such an unexplored topic required the questions to be open-ended.

Following this, the *data analysis* step was performed by coding the observations from the collected data into the taxonomy. Here, an initial taxonomy had been developed based on concepts from the existing DevOps methods for web development, and the new taxonomy was generated by modifying this template based on the collected data.

Finally, a *conceptual evaluation* was performed in order to ensure that the final taxonomy encompasses all the relevant concepts. The taxonomy was validated by involving additional researchers and practitioners from both of the industrial case studies. As suggested by Ralph [Ral18], the participants were asked to identify weaknesses in the theory, as well as to define the evaluation criteria to determine whether the taxonomy is credible and transferable.

5.3 Case Studies

This section describes the two industrial case studies employed to define and evaluate the taxonomy. The first case study belongs to the vertical transportation domain (i.e., elevation), and the second one is from the railway transportation domain. In what follows, we describe the relevant subsystems and software development processes that are currently employed in these companies, as well as the expected benefits that would be provided by the adoption of DevOps methodologies into their current workflow.

5.3.1 Elevation Domain

Orona, one of the leading elevator companies in Europe, works on the design, manufacturing, installation, maintenance, and upgrading of elevators, escalators, moving ramps, and walkways.

A multi-elevator installation is a complex distributed CPS with various components interacting with each other in order to provide the best possible Quality of Service (QoS) for its passengers. This QoS can be measured by employing domain-specific metrics such as the Average Waiting Time (AWT), and more recently, additional criteria such as the energy consumption. Most of the changing functionalities of the elevators, including the optimization of QoS metrics, is provided by the software controlling the elevator installations.

An overview of the development process at Orona was previously shown in Figure 4.3, and the system's architecture is described in Section 4.4.1. As a reminder, the *traffic master* is the component which manages the passenger flow of the installation, and includes software modules such as the elevator dispatcher algorithm and the access control. The traffic master is constantly evolving in order to improve functionality, fix bugs, or adapt to new requirements and regulations. Currently, as depicted in Figure

4.3, the deployment of a new version of this software requires it to be validated at the SiL and HiL levels, as well as a deployment and test at the elevator installation location, performed manually by a technician.

Given this complicated process, and the several manual and semi-manual steps involved, this product would greatly benefit from DevOps methodologies to streamline them. More specifically, the transition to the Hardware-in-the-Loop (HiL) phase requires hour of work in order to ensure a proper configuration by domain experts, and the test executions at this level also currently require manual supervision. Additionally, the deployment of new software versions, currently a manual task, could also be performed remotely, combined with a set of automatic checks of the new software's functionality. On the other hand, the monitorization of the installation and the subsequent reproduction of the relevant scenarios at the SiL phase is also a complex and error-prone process which complicates the identification of bugs or possible optimizations. With DevOps methodologies, these tasks could be further automated, resulting in an increase of the software quality and a reduction in time to deploy changes.

5.3.2 Railway Domain

Bombardier Transportation (BT) is a leading company from the rail industry which manufactures, maintains and manages rolling stock. Its products include light rail vehicles, metros, and commuter and regional trains.

Train control systems are distributed CPSs with many components, such as displays, buttons, or engine control systems. The Train Control and Management System (TCMS) is the core of this system, and is involved in most of the functionalities of the train, either for control or monitoring. As a component with many complex software modules, it is prone to requiring updates to add new functionality or reconfigure or fix existing modules.

The *test plan* documents the testing process of the system, including scope, approach and resource allocation. It also describes the PASS/FAIL criteria for each of the tests, which is linked to one or more of the system requirements. All the software development processes (including requirements engineering and testing) are performed according to safety standards and regulations [CEN01]. Figure 5.1 depicts the software development process at BT. As shown in the figure, the testing is performed at three different levels: (1) software component test, (2) function test, and (3) system test. Component and function tests are generally executed using Model-in-the-Loop (MiL) or Software-in-the-Loop (SiL) simulators, while system tests are typically performed on a HiL environment.

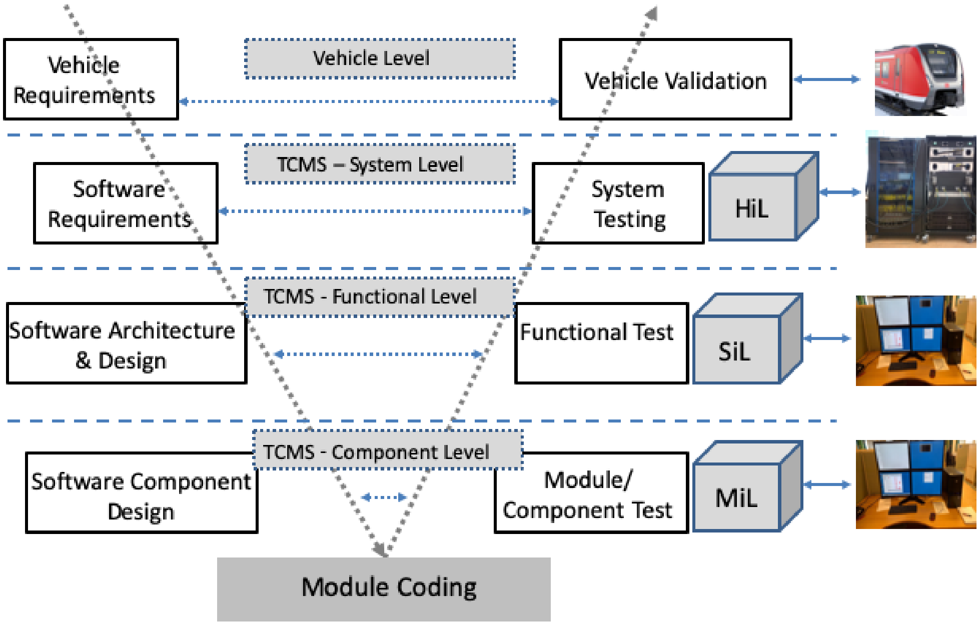


Figure 5.1: Software Development Process at BT [AGA+20]

The execution of test plan at all levels follows the same general procedure: (1) prepare test infrastructure, (2) develop software components/features, (3) implement test cases, (4) build and deploy on test bench, (5) execute test cases, (6) record failures (if any), (7) generate test report, (8) release software when no test failures remain. Many test activities, such as setting up the test environment, currently require manual intervention.

The adoption of DevOps methodologies is expected to allow the efficient management of different configurations and requirements across the different testing levels, resulting in a faster and less error-prone process. Similarly to the other case study, the deployment of new software and monitoring of the installation are also tasks that are planned to be automated. The information collected by the monitoring is also expected to enable the generation of better test cases for subsequent testing tasks.

5.4 Taxonomy for CPS DevOps

There are two main classification approaches to organize a taxonomy: enumerative or faceted [RH17]. An *enumerative* taxonomy contains a predefined set of classes, resulting in very simple schemes that are appropriate for well-explored domains with an established and complete knowledge base. On the other hand, a *faceted* taxonomy contains multiple dimensions for the classifications, resulting in more flexible, yet complex structures. The latter is considered more appropriate for new domains which are not yet well-established [UBBM17], and we therefore chose the faceted approach for our new taxonomy.

The final taxonomy consists of four orthogonal facets, one of which contains multiple levels of sub-categories. The full taxonomy is shown in Figure 5.2.

5.4.1 Lifecycle stage

This facet of the taxonomy represents the X-in-the-loop system execution level to which the requirement applies to. While some requirements may be specific to one of these levels, many of them will be applicable to multiple or even all of them. This is an aspect specific to CPS development which is not present in other software-related domains, and thus, it is not something considered by current DevOps systems. As shown in Figure 5.2, our taxonomy considers four different classes, which are the ones relevant to our case studies, and the most common ones in our experience. Nevertheless, it is true that more execution levels might be needed for other case studies, such as Processor-in-the-Loop (PiL). Given the innovative nature of this work, we consider our taxonomy a template which can be expanded in the future depending on the needs of other users from different domains, such as adding new lifecycle stages that are not considered in our current taxonomy.

The first X-in-the-loop execution level we consider is **MiL**, where the entirety of the system is defined and executed in a modelling environment, such as Modelica [FPA$^+$20, 216] or Simulink [DVDBSR13, Mat22]. Such an environment allows an easy definition, analysis and debugging of heterogeneous systems (environment, hardware, and software), which allows an early and efficient detection of many types of failures. Nevertheless, there are many potential issues that cannot be revealed at this level, such as those related with communications between components or problems that can only be observed in real-time execution environments.

Next, the **SiL** level consists in running the real CPS software on a development machine and a simulated environment. Using the real software allows the detection of additional problems which could not be observed at MiL, such as numeric precision issues. However, many aspects of the hardware and the environment are still not testable, and the fact that the software is not running in the target processor means that even some software-related issues can still be missed.

Following that, the **HiL** execution level consists in deploying the software on the real hardware, but some controlled environment. For instance, in the elevation domain, the real computational unit, buttons and indicators are used, as well as the real Controller Area Network (CAN) bus for communication between them, but the

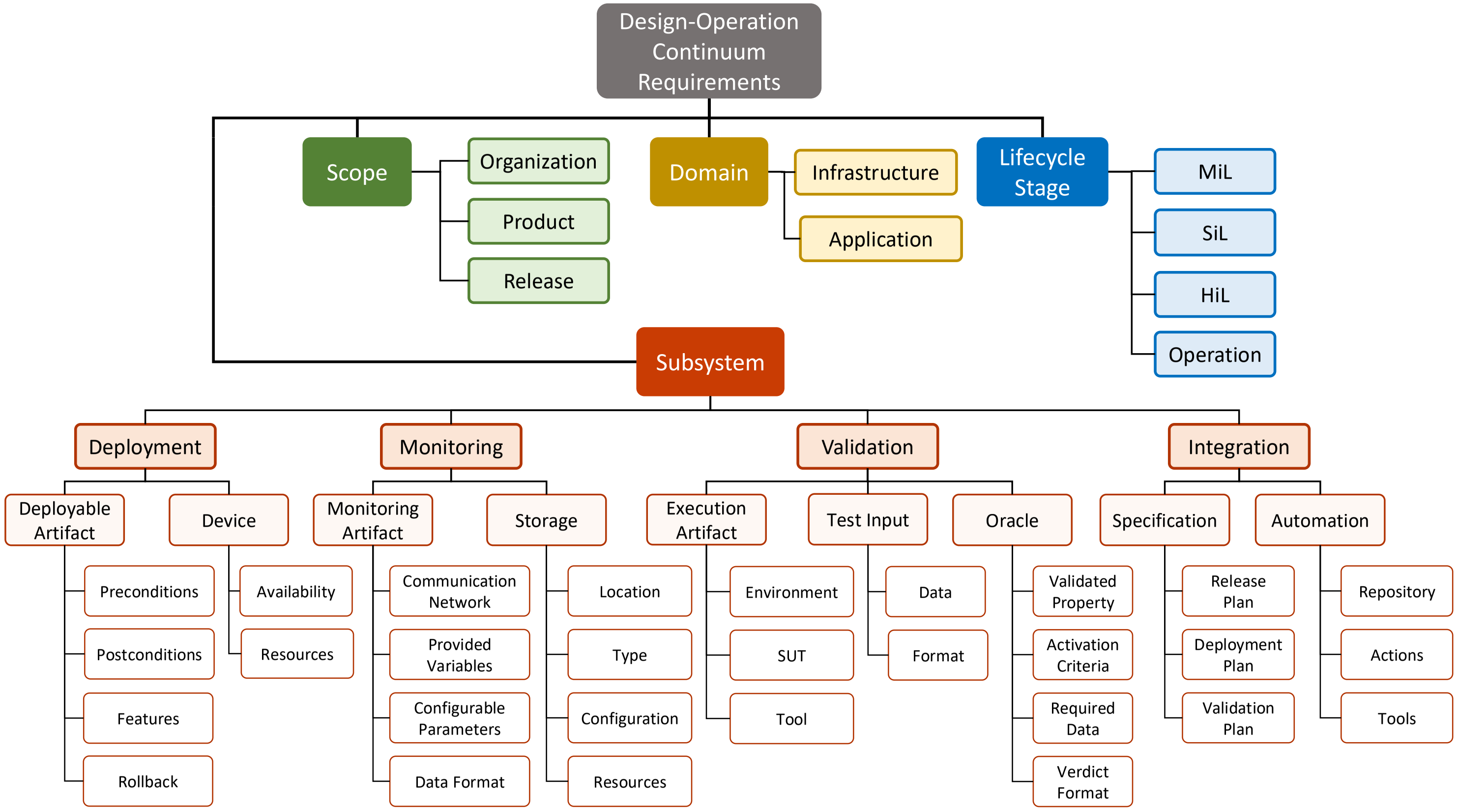


Figure 5.2: Taxonomy of DevOps Requirements for CPSs [AGA^{+}20]

elevators and the shaft are miniature models. At this level, the system executions are real-time, which allows the detection of many new error classes, but also makes executions much more costly.

Finally, the **Operation** execution level corresponds with the real environment in which the CPS is supposed to run, possibly the production environment. The system will be running real scenarios at this stage, so executing specific test cases will no longer be possible. Nevertheless, some non-intrusive verification techniques, such as runtime verification [LS09, BFFR18], can still be performed.

5.4.2 Scope

The scope defines the applicability of a requirement, and is intended to specify and enable its reusability. The three classes we define in this taxonomy are generic, and may have to be adjusted or extended for each organization depending on the strategy used to define products.

The **organization** scope refers to DevOps requirements that are applicable throughout all the products from the organization, and corresponds with the most general and reusable requirements. For example, this may include being able to deploy files through a SSH connection, if that is the method that will be used to deploy software updates throughout all the organization's products.

Next, the **product** scope refers to more specific requirements that only affect a particular product line, but can be reused across the different configurations or variants of the same product. As an example from the elevation domain, for elevator dispatchers, the DevOps system should be able to deploy and launch the domain-specific simulator used to test the product at the SiL execution level.

Finally, the **release** scope indicates that the requirement is specific to a particular product variant or configuration. For instance, the verification of the access control feature of an elevator dispatcher will be specific to the installation where such a security feature has been requested by the customer.

5.4.3 Domain

The domain of a requirement is a facet intended to facilitate its assignment to the appropriate team within the organization. We currently envision two categories, the first of which would be handled by DevOps specialists (could be the organization's IT department), whereas the second one would be assigned to the domain experts (i.e., the product development team).

The first domain class is **infrastructure**, which encompasses requirements for the DevOps infrastructure itself. As an example, these might include monitoring the status of the deployment process. The roles responsible for these requirements do not need to have knowledge of the CPS under development, but should ideally be knowledgeable of DevOps systems instead.

The other domain we propose is **application**, which encompasses requirements related to the product under development (i.e., the CPS). For instance, these requirements might include system monitoring, such as tracking the status of the elevators installation based on CAN messages between components. These requirements would be best defined and managed by the same domain experts who develop the product, since this is the group which possesses the relevant knowledge.

5.4.4 Subsystem

The subsystem facet defines the DevOps task for which the requirement is relevant. Our version of the taxonomy specifies four DevOps tasks: Deployment, Monitoring, Validation and Integration.

The **deployment** subsystem allows uploading new software versions and other files to a target machine [NFE$^+$19], either in the real CPS in production or in a development environment (e.g., for testing). The deployable artifacts include not only components related to the CPS, but also components of the DevOps system itself, such as monitors or test oracles.

The **monitoring** subsystem extracts data from the CPS so that it can be analysed [LMLK14]. This analysis may be done by other subsystems for tasks such as validation, failure reproduction, or predictive maintenance. Furthermore, the data can also be stored to perform offline analyses later on.

The **validation** subsystem provides various techniques to perform testing and other verification and validation activities, such as different types of test oracles. This subsystem also encompasses the execution of the CPS in a controlled environment for testing activities, such as launching a simulation.

Finally, the **integration** subsystem manages the interaction between multiple subsystems in order to have them perform useful tasks jointly. For instance, the test oracles used for validation typically require monitoring the system and collecting the necessary execution data, and these DevOps components will also need to be deployed with all their dependencies before they can start running. We define an integration subsystem which will streamline the integration of the other subsystems in order to facilitate the full automation of the DevOps pipeline.

Next, we describe the validation subsystem in more detail, as this is the DevOps aspect that is directly relevant to this thesis.

Validation Subsystem

Testing, verification and validation activity requirements are all encompassed by this subsystem. This task is particularly important in CPSs, since they are often safety-critical or have failure modes which can have severe consequences. This is also the subsystem to which the work of this thesis will contribute to, and where the proposed verification methods or techniques will be implemented. Recall that in the context of CPS development, there are multiple execution levels where verification tasks can be performed, with testing the software in the real CPS (Operation) usually being the last and most costly environment. Before that, the software is typically verified in multiple MiL, SiL and HiL setups, where various simulator tools and test benches are used to execute test cases. In order to facilitate a full automation of such a multi-level verification process, the test oracles which encode the various requirements of the CPS should be reusable at all the execution levels, including Operation.

We consider three sub-classes of requirements for this subsystem: Those related to the artifacts required to execute the CPS, the requirements related to the test inputs, and those referring to the test oracles.

Firstly, the *execution artifact* class refers to the requirements related to the system execution itself. Here, we distinguish the following sub-categories: (1) *Environment* conditions in which the System Under Test (SUT) is executed (e.g. the elevators installation shall have 10 floors). (2) The requirements for the *SUT* itself (e.g. the elevators dispatcher shall be compiled into a library which can interface with the SiL simulator). (3) *Tool* requirements, encompassing simulation or modelling environments required to execute the system (e.g. the Elevate simulator shall be used to execute test cases).

Secondly, the *test input* class is related with the requirements for the data which will be injected to the SUT in order to test it. We distinguish the following sub-categories: (1) Test input *data* related requirements (e.g. at least one test input must have concurrent passenger calls). (2) *Format* of the test inputs (e.g. the test inputs will be provided as a XML file compatible with the Elevate simulator).

Thirdly, the *test oracle* class is related with the components which determine whether the SUTs execution has revealed a failure or the behaviour was correct. When the SUT is executed, the monitoring subsystem will record execution data, which can be consumed by a test oracle in order to emit a PASS or FAIL verdict. Note, however, that the monitoring data may be used for purposes other than verification, which is why monitoring is classified as a different subsystem. This class is further divided into: (1) *Validated property*, referring to the system's requirements which are checked by the test oracle (e.g. the AWT of the installation shall not be higher than 30 seconds). (2) *Activation criteria*, which defines the conditions that must be met in order to consider the test oracle's verdict valid or useful (e.g. the checked property shall hold at any point starting from a minute after the execution starts until the execution finishes). (3) *Required data*, defining the data fields that the test oracle needs to receive from the monitoring system (e.g. the oracle shall be provided with the AWT of the installation of every 5 minute window). (4) *Verdict format*, the syntax and semantics of the verdicts provided by the test oracle (e.g. the test oracle shall provide a quantitative verdict from 0 to 1, with 1 meaning full compliance and smaller values meaning a smaller degree of compliance).

5.5 Related Work

There are many previous works in the literature which have developed taxonomies for complex systems. Roman et al. [Rom85] presented a classification of requirements specification techniques, highlighting the role they have in the development cycle and the common problems that are faced. Jarke et al. [JBR+93] describe an ontology of requirements engineering with three different worlds: (1) *Subject world* (representation of the domain), (2) *Usage world* (interfaces with the users), and (3) *Development world* (evolution process of the system). Hughes et al. [HRS94] proposed a taxonomy for requirements analysis with two dimensions: (1) *Concerns*, which relate to the views of the stakeholders and (2) *frames*, which represent the views of technical specialists. Nuseibeh et al. [NKF94] propose a viewpoint interaction model to reconcile multiple partial requirements specifications with different representation schemes and development strategies, since CPSs are usually developed by multiple participants with different requirements that may be complimentary, overlapping, or even contradictory with each other. White et al. [WE95] defined a taxonomy for specifying complex system requirements and specifications, including non-functional aspects and specification for the growth and change of the system. They also present a classification for different specification approaches, ranging from informal (natural language) to formal (mathematical). Hasan et al. [HLN14] documented the techniques for specifying non-functional requirements that exist in the literature and performed a qualitative analysis of their scopes and characteristics.

Regarding requirement elicitation approaches for CPSs, Reza et al. [RKS+16] elicited quality-based requirements, such as those related to availability or performance, for a CubeSat spacecraft system. Wiesner et al. [WHHT16] present a gamified approach for eliciting stakeholder requirements, which they successfully applied in three industrial CPS use cases. Loucopoulos et al. [LKC19, LKM22] introduce the e-CORE (early Capability Oriented Requirements Engineering) framework, an asset-centric approach for the traceable requirements engineering of Cyber-Physical Production Systems (CPPSs), which they apply to an industrial case study from the automotive domain.

To the best of our knowledge, this is the first taxonomy for the requirements of a DevOps system for CPSs. Due to the particular challenges that this specific type of systems pose, the requirements will be significantly different compared to other domains where DevOps systems have already been implemented, such as web development. Thus, we have generated a new taxonomy based on the requirements from two industrial systems belonging to different domains.

5.6 Conclusion and Future Work

This chapter describes the development of a taxonomy which can be used to elicit requirements for CPS DevOps systems. The taxonomy has been developed based on two industrial CPS case studies from different domains, following the guidelines proposed by Ralph [Ral18]. Experts from both case studies have been interviewed in order to develop and validate the taxonomy, and a set of requirements have been instantiated for each of the case studies based on it.

The current taxonomy is intended to be used as a template that can be extended or modified for other domains, where the desired scope or the requirements for a DevOps system might be significantly different to our current case studies. In the future, a more comprehensive version of the taxonomy may be published based on additional sources from the literature and experiences from other industrial case studies. The *Subsystem* facet of the taxonomy might also be extended in order to support additional functionalities, such as unforeseen situation detection or automated fault recovery.

Chapter 6

Microservices for CPS DevOps

Considering the requirements for a Cyber-Physical System (CPS) DevOps pipeline we defined in Chapter 5, this chapter presents a domain-agnostic DevOps architecture for CPSs based on microservices [GAA$^+$21b].

This work has been developed in a collaboration between Ikerlan, Mondragon University, Simula Research Laboratory, and Orona [GAA$^+$21b], in the context of the Adeptness project [ade]. Same as for Chapter 5, the work for this thesis has been mainly focused on the validation subsystem of the DevOps architecture, with some general aspects developed jointly.

6.1 Introduction

CPSs integrate software with hardware components and the physical environment [Alu15]. The software of these systems, which implements an increasing amount of their functionality, needs to evolve constantly in order to keep up with bug fixes and new requirements [AGA$^+$20]. In order to reduce the cost and increase the quality of the software development process, the adoption of DevOps techniques in the context of CPS development has been proposed in Chapter 5. In that chapter, the groundwork for eliciting the requirements for such techniques is laid down, and the requirements for two industrial CPSs are elicited.

This chapter presents an architecture for CPS DevOps based on microservices, which has been instantiated for the elevation case study described in Chapter 5. The current processes employed by Orona, the company developing the elevator systems, still rely on manual intervention in many instances: The software deployment and validation steps are still semi-manual, and there is a lack automatic feedback mechanisms that prevent the accurate replication of bugs observed during operation in a controlled environment. The architecture presented in this chapter supports automatic deployment, continuous monitoring of the installation, and fully automated validation. This chapter will focus mainly on the validation subsystem, which is the

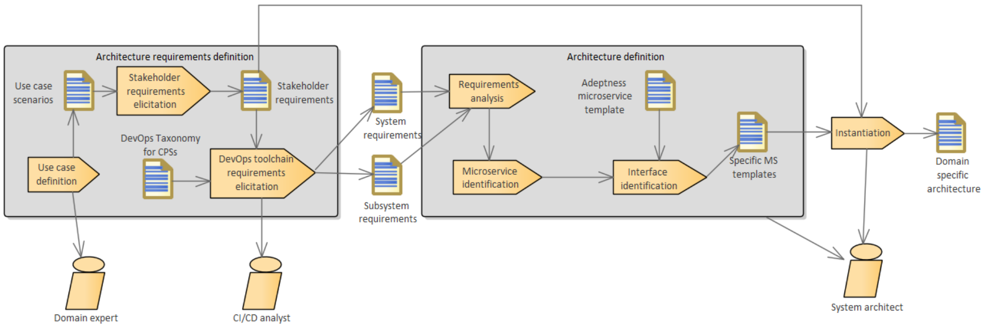


Figure 6.1: Methodology for developing the DevOps microservice architecture [GAA+21b]

one where we will integrate the techniques developed in this thesis.

The rest of this chapter is structured as follows: Section 6.2 describes the method used to develop the architecture, Section 6.3 architecture itself, Section 6.4 presents the prototypical implementation and its evaluation, Section 6.5 discusses the related work, and Section 6.6 concludes the chapter and highlights future work in this line of research.

6.2 Architecture Development Methodology

We first defined a systematic methodology to develop the DevOps architecture, including the definition of the requirements and the technical decisions made to satisfy them. Figure 6.1 shows an overview of the full process, including the requirements elicitation and the definition of the architecture itself.

An architecture based on microservices was opted for due to its advantages in terms of complexity management and error-proneness [OEC17, GL18]. We specifically considered the fact that the DevOps system would be implemented incrementally, and that extensions and changes are expected to be introduced regularly due to the novelty of DevOps toolchains in the domain of CPSs.

Thus, the architecture development methodology comprised the following steps:

1. **Requirements definition**

   1.1. **Use case definition.** First of all, the use-case scenarios for Orona were defined by domain experts. These scenarios are accessible in [Ikeb].

   1.2. **Stakeholder requirements elicitation.** After defining the use-case scenarios, a domain expert elicited a total of 56 requirements from the stakeholders. The requirements can be found in [Ikea].

   1.3. **Toolchain requirements elicitation.** Using the taxonomy developed in Chapter 5 [AGA+20], the stakeholder requirements were classified into two main categories: (1) System requirements for the overall architecture, and (2) the requirements specific to the individual subsystems. These requirements, along with the test cases to validate them, can be found in [Ikea].

2. **Architecture development**

   2.1. **Requirements analysis.** Using the requirements elicited in the previous steps, the technical requirements for the architecture were defined by a system architect.

   2.2. **Microservice identification.** A series of microservices were identified, and the previously defined requirements were assigned to each of them.

   2.3. **Interface identification.** On the one hand, a single high level interface for all the microservices was defined for both C and Python implementations. On the other hand, the specific interface for each of the microservices was defined in the form of synchronous and asynchronous messaging protocols.

3. **Instantiation.** As a final step, the microservice templates were instantiated and integrated together.

6.3 Microservices Architecture

The proposed architecture comprises several services which can be composed together and reused across different lifecycle stages when possible. Each microservice is responsible for a single specific task within the CPS development and operation process, and provides different communication protocols to interact with it. This design ensures flexibility in terms of both extending and scaling the system, which might be necessary due to the heterogeneous nature of CPS ecosystems and their ever-changing requirements [ade].

All the microservices defined in this architecture provide a basic common interface, which includes endpoints for synchronous (HTTP) and asynchronous (MQTT) communication. The synchronous interfaces allow getting information about or controlling a specific microservice, whereas the asynchronous interfaces can be use to discover the available microservices. All of these common interfaces are part of the basic microservice template that has been developed [ade].

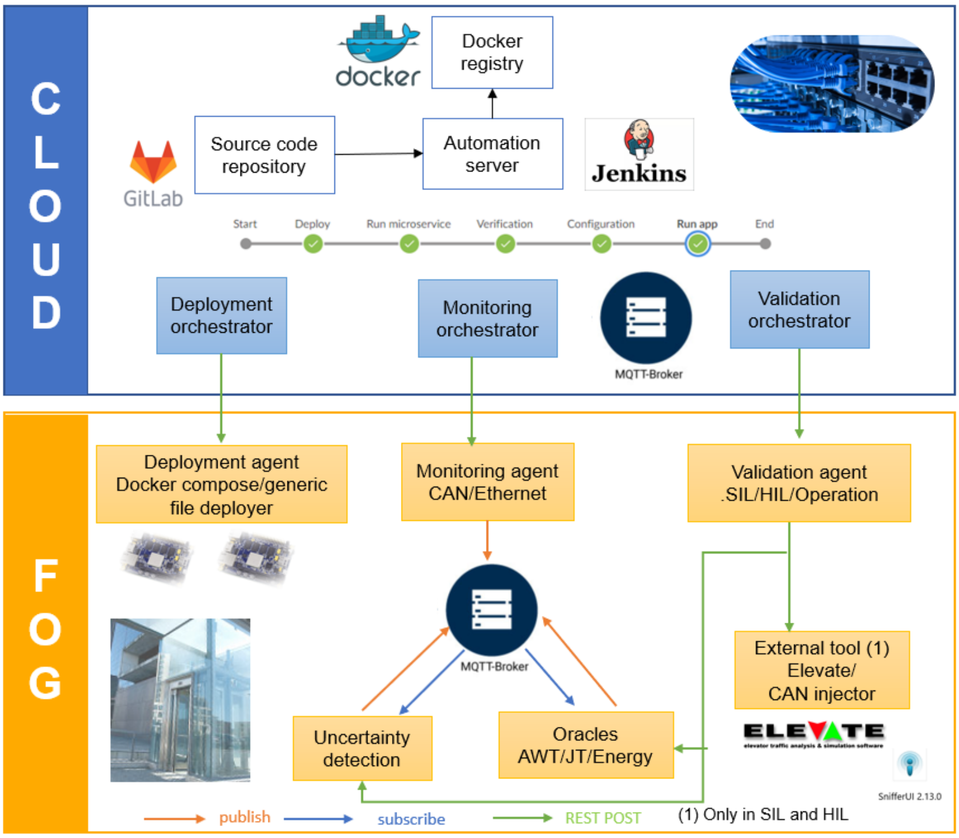


Figure 6.2: Overview of the DevOps microservice architecture [GAA+21b]

6.3.1 Subsystems

Figure 6.2 shows an overview of the architecture instantiated for Orona's systems. The DevOps subsystems developed for this case study are: *Deployment*, *monitoring*, and *validation*, as well as an *automation* server to orchestrate the tasks to be performed by the rest of the subsystems.

The **automation server** is the subsystem which performs the programmed DevOps tasks by orchestrating other subsystems. It also monitors the source code repositories in order to trigger specific plans when a new commit is made. When a new plan is launched, the required artifacts are generated, Docker images are created and stored in the Docker registry, and the configurations or plans for the individuals subsystems are pushed so that they start working and coordinate with each other. As shown in Figure 6.2, the automation subsystem is executed exclusively in the cloud.

The **deployment subsystem** is responsible for moving the artifacts required for a specific task to their target nodes. This subsystem comprises the deployment orchestrator (cloud) and the deployment agent (target device).

The **monitoring subsystem** provides information about operational variables of the CPS, possibly in real-time. It provides this information to other subsystems which may need it, such as the oracles from the validation subsystem. This subsystem comprises the monitoring orchestrator (cloud) and the monitoring agent (target device).

Finally, the **validation subsystem** validates the system's behaviour and reports to other subsystems when a potential failure is identified. We now describe this subsystem in more detail.

Validation Subsystem

The continuous validation subsystem performs verification and validation activities at all the X-in-the-loop execution levels, including Operation. At the Model-in-the-Loop (MiL) and Software-in-the-Loop (SiL) levels, these activities can be executed using only software and generic computational resources (with a tool such as Elevate [Lim22] simulating the physical parts of the system), whereas Hardware-in-the-Loop (HiL) and Operation require setting up the physical components of the system and interacting with them. This subsystem currently comprises five different microservices:

- *Validation orchestrator*: This microservice receives the validation plan from the automation server and executes it. This orchestrator communicates with the validation agents from different nodes in order to perform the validation tasks. A validation plan may contain tasks which require multiple execution levels. This microservice is intended to be executed in the cloud.
- *Validation agent*: These agents manage the validation tasks for a node. Their tasks include launching the executions (e.g. launching the simulator tool at the SiL level), activating the test oracles, and providing the oracle verdicts to the validation orchestrator. Most of these tasks are handled by the next three microservices, which are instantiated by this microservice. The agents are executed in the edge nodes.
- *External tool*: This microservice launches domain-specific tools which are required to execute the CPS at the MiL, SiL, or HiL levels. In Orona's case, two different external tool microservices have been defined:
    - *Elevate launcher*: The domain-specific simulator required for SiL executions.
    - *Controller Area Network (CAN) frames injector*: A tool to simulate scenarios at the HiL level by injecting frames to the system's CAN bus, such as simulated passenger calls.

- *Oracle*: These microservices subscribe to the operational data published by the monitoring subsystem and verifies the correctness of the CPS's behaviour. In the case of Orona, many of the oracles are based on the Quality of Service (QoS) measures from the system, such as the Average Waiting Time (AWT) of the passengers or the energy consumption of the elevators installation. Some special types of oracles have also been implemented, such as the metamorphic test oracles developed in this thesis [ASA$^+$20, AVS$^+$22a], or others that employ machine learning [AAI$^+$21, GAA$^+$22].

- *Uncertainty detection*: This microservice employs machine learning techniques in order to detect unforeseen situations in the system by combining operational data and other information collected during development, such as test logs.

6.4 Implementation and Evaluation

In order to evaluate this architecture, a prototype implementation has been developed for Orona's SiL validation use case[1]. The prototype consists of a Jenkins pipeline which incorporates all the subsystems described in Section 6.3. All the microservices are provided as Docker images which have been pushed to a Docker registry.

The deployment subsystem continuously queries the repositories, and launches the deployment pipeline when a change is made. An agent fetches and launches the Docker images of the microservices, making sure that they are started correctly. The monitoring pipeline configures the monitored variables and the corresponding topics where their values will be published, and then activates the monitors so they start publishing operational data. When all the monitors have been activated successfully, the validation pipeline is started.

The validation pipeline configures the topics for the operational variables obtained by the monitoring subsystem, as well as the topics where the oracle verdicts will be published. Then, the microservices are activated, and the test oracles are continuously polled until a verdict is issued. The verdicts will then be published to the corresponding MQTT topics so that the success or failure of the validation can be reported.

If any failure happens during any step of this DevOps pipeline, the process is aborted, all the Docker containers are stopped, the environment is cleared, and the error is logged.

The work of this thesis is being developed in the context of the validation subsystem of this architecture. The following are the benefits that this approach can bring to Orona's current verification workflow:

[1] Video of the prototype: `https://youtu.be/uoq9n9k4kgc`

- While the unitary test cases of short scenarios will still be defined manually, the generation of longer test cases will be based on real data from the installation, rather than theoretical passenger profiles. The new test cases will be more likely to reproduce real problems.
- The configuration of the validation processes will be based on pre-existing templates rather than done from scratch. The execution of the validation processes will be triggered automatically from Jenkins rather than manually by the developers.
- Automatic and reusable test oracles will be deployed throughout all the execution levels, increasing the number of bugs detected and reducing efforts. Previously, human oracles where used extensively, which are costly and error-prone.
- The bugs identified during execution will be reproducible in the development environment using the data collected by the monitoring subsystem. Currently, logs are enabled on demand and inspected manually, which is costly and less accurate.

6.5 Related Work

Microservice-based architectures are already being adopted in CPS and Internet of Things (IoT) domains due to their flexibility, maintainability and scalability [BGT16]. Our architecture is designed to obtain these benefits for integrating DevOps processes for CPS development.

Other works have proposed microservice architectures for the CPS itself. Alam et al. propose an architecture based on Docker in order to develop highly modular systems and simplify their management, and conclude that their approach ensures the reliability and recoverability of the system, even for time sensitive applications [ARF+18]. Thramboulidis et al. describe a framework for exploiting microservice architectures in the domain of manufacturing systems, and weigh their pros (e.g. flexibility) and cons (e.g. the high latency introduced by the microservice containers and their integration protocols) [TVS18]. In contrast, our work adopts microservices for development and operational tasks, rather than for the system itself.

Due to the long development life-cycles of CPSs, the adoption of DevOps methods in this domain is being studied in order to speed up the related processes [ABG+20]. To this end, different techniques have been proposed in the literature, such as model-based engineering [CW19] or Digital Twins [UQES20]. Our work proposes a microservice-based architecture to solve this problem. Most of the proposed methodologies and tools focus exclusively on specific DevOps activities, such as deployment [RVT+18, FNS+19, PT20], monitoring [VCHB+18, MRB19], or validation

[BSB+16, MLM18]. In contrast, our architecture integrates all of these subsystems, and is designed to be extended if more functionality is needed.

There have also been works which apply microservice architectures for DevOps activities in the domain of cloud infrastructure management. Kang et al. propose a container-based approach to redesign the OpenStack deployment architecture, and discuss the benefits and limitations [KLT16]. Miglierina et al. present a factory pattern for automatically composing and configuring monitoring systems, which is intended to reduce the entry-cost of adopting monitoring solutions [MT17].

6.6 Conclusion and Future Work

This chapter describes a microservices based architecture for enabling DevOps methods in the domain of CPSs, and also presents a prototype implementation for Orona's SiL system verification use case. This architecture is based on the requirements elicited following the work from Chapter 5.

The adoption of DevOps in the domain of CPSs, where the development lifecycles are so long and the tasks are so fragmented, has the potential to greatly reduce the cost and increase the quality of these systems. However, due to the heterogeneous environments used, as well as the different needs of each CPS, it is difficult to formulate a generic DevOps system which can easily be adapted to all of them. Our proposed microservice architecture provides the modularity and flexibility required to enable the maximum reuse of the subsystem artifacts, while ensuring that new or modified services can be integrated as easily as possible in order to satisfy the specific requirements for each CPS.

In the future, this architecture will be refined and extended in order to accommodate more of Orona's use cases. For instance, automatic recovery mechanisms are planned to be integrated, which will support scenarios such as rolling back a software release if a significant performance regression is detected during online verification.

Part III

Oracle Problem

Chapter 7

Manual Generation of Metamorphic Relations

Within the context of Cyber-Physical System (CPS) development methodologies, Part II presented a general DevOps taxonomy and architecture proposal in order to improve the quality and reduce the cost of developing these systems. This part focuses on the more specific goal of this thesis, which is the mitigation of the test oracle problem in CPSs. This is currently one of the major bottlenecks for the development of reliable systems in a cost-effective manner.

This chapter presents a solution to the oracle problem in CPSs based on a technique called Metamorphic Testing (MT). More specifically, we research the cost-effectiveness of employing this technique for determining whether the performance or Quality of Service (QoS) measures obtained by a system are adequate or reveal a system failure. Our research reports the results from applying this technique on systems where the only current test oracles are regression tests and manual decisions, and presents Metamorphic Relation Patterns (MRPs) and lessons learned which might help the adoption of MT for CPS verification [ASA$^+$20, AVS$^+$22a].

The effectiveness of MT is then compared with regression oracles, which is another solution that is being researched on the same industrial case study from the elevation domain that we use to evaluate MT [AAI$^+$21, GAA$^+$21a, GAA$^+$22].

7.1 Introduction

Cyber-Physical Systems (CPSs) are highly complex systems that often have strict safety and robustness requirements, and their compliance must be ensured through a thorough verification and validation process. Indeed, verification is one of the major concerns when developing these systems, and is even estimated to consume over 50% of the development resources in some domains, such as aviation [BG11].

One of the main challenges in this context is deciding whether the system's behaviour on a test case is valid or not. In performance testing, for instance, the exact frontier between correct and incorrect values for a performance metric is often unknown. This inability to determine whether a test outcome is correct or not is known as the *test oracle problem*, and is widely recognized as one of the fundamental problems in software testing [Wey82, BHM$^+$14]. Currently, many industrial practitioners resort to employing *human oracles*, i.e., having test engineers manually assess the outcome of a test case [FJA$^+$15, ASA$^+$20]. Unfortunately, this solution is costly and does not scale well as these system's complexity increases, since it prevents the full automation of verification processes.

Metamorphic Testing (MT) is an alternative approach to testing which consists of comparing the outcomes from multiple test executions with known *input relations* in order to identify violations of *output relations*, which are necessary properties that must be held between test executions [CCY98]. These relations among the inputs and outputs of multiple test executions are known as Metamorphic Relations (MRs). For example, consider a self-driving vehicle traversing a given route, and the time it needs to reach the destination as a performance metric. With traditional testing, it would be difficult to determine if a time to destination obtained by the system ($time(R)$) is acceptable for the route R, since there are many factors to consider, such as the vehicle parameters or the road topology. However, if we were to reverse the original route, such that the vehicle would have to traverse the same path backwards, we would expect the time to destination to be very similar to that of the original route. Formally, we can define the output relation of this MR as: $time(R') \simeq time(R)$, and the input relation as: $R' = reverse(R)$.

It is often possible to define abstract MRs, which represent many possible MRs that can be instantiated for specific contexts. These abstract MRs can facilitate the adoption of MT in new domains, since they provide patterns that can be referenced in order to identify new MRs for those systems, which is a necessary step of MT. These abstract MRs are called Metamorphic Relation Pattern (MRP) [SPTRC18, ZSCT18]. It is also common to define patterns where only the input or the output relations are specified, which are called Metamorphic Relation Input Patterns (MRIPs) [ZSCT18] and Metamorphic Relation Output Patterns (MROPs) [SPTRC18] respectively.

This chapter presents the Performance Variation (PV) pattern, a MRP which can be used to identify failures in CPSs by defining MRs based on input changes with predictable impacts in performance [AVS$^+$22a]. Using this pattern, we instantiate MRs for the contexts of testing Autonomous Driving Systems (ADSs) [Val21] and elevation systems [ASA$^+$20], and our empirical evaluation demonstrates that this approach is effective at detecting failures automatically based on performance metrics. Furthermore, we compare the obtained results with threshold-based and regression oracles [AAI+21, GAA+22], and conclude that our MRs provide clear advantages over these alternatives in terms of cost-effectiveness. We also make the replication package for the open-source ADS case study publicly available [AVS+22b].

The rest of this chapter is structured as follows: Section 7.2 presents the PV pattern with various examples, Section 7.3 defines the experimental evaluation and discusses the obtained results, Section 7.4 points out the threats to validity, Section 7.5 describes the related work, and Section 7.6 concludes the chapter.

7.2 Performance Variation Pattern

This section describes PV, the MRP which encompasses all the MRs employed in this chapter. We define this MRP as follows [AVS+22a]:

Performance Variation (PV). This pattern represents those MRs that involve a change in the source input that has a predictable effect on the performance of the test case execution.

Even though most of the research on MT has been focused on functional fault detection [SFSRC16, CKL+18] so far, more recent work has started applying *performance metamorphic testing* [STDRC17, STDRC18] in various contexts. This type of MT defines relations based on the performance metrics of the system, such as execution time or energy consumption. For instance, if we have an algorithm which applies a filter to an image, we can intuitively assert that applying the filter with an upscaled version of an image should require more execution time than the original, since the upscaled image has more pixels to process: $T(filter(I)) \leq T(filter(upscale(I)))$.

Performance measurements are inherently non-deterministic. In many systems, external factors such as the system workload or environmental changes can affect them in unpredictable manners, and the measurements themselves are often not completely accurate. Many systems are also non-deterministic themselves, and may behave differently even if same inputs are provided. The presence of so many possible sources of noise means that it is not feasible to perform a direct comparison between the performance measurements obtained from different system executions (e.g., expecting identical execution times when running the same test twice). Several approaches have been proposed to address this issue, such as using tolerance thresholds when comparing performance measurements [MSK09a] or comparing statistical distributions from multiple runs rather than individual measurements [GM07]. Thus, when we define our MRs, we refer to a performance measurement being *lower* ($\lesssim$), *higher* ($\gtrsim$) or *similar*

($\simeq$) than another, and we assume that some of the previous methods will be used to mitigate the noise in the measurements.

In what follows, we present several MRIPs derived from the PV pattern, each of which represents groups of MRs sharing the same input relation, as well as at least one MR that can be instantiated for a specific performance metric. Note that these are examples provided for illustrative purposes, and are therefore rather simple. Section 7.3 will present the MRs implemented in practice, and Section 7.3.4 will discuss the specific reasons why some MRs were not used or implemented differently.

7.2.1 Elevation Control System

Passenger elevator control systems fulfil vertical transportation requests by coordinating one or more elevators so that all the passengers are attended as efficiently as possible. The performance or Quality of Service (QoS) of these systems can be measured with various metrics, such as the total execution time for all the requests, the average waiting time of the passengers, or the energy consumption of the elevators.

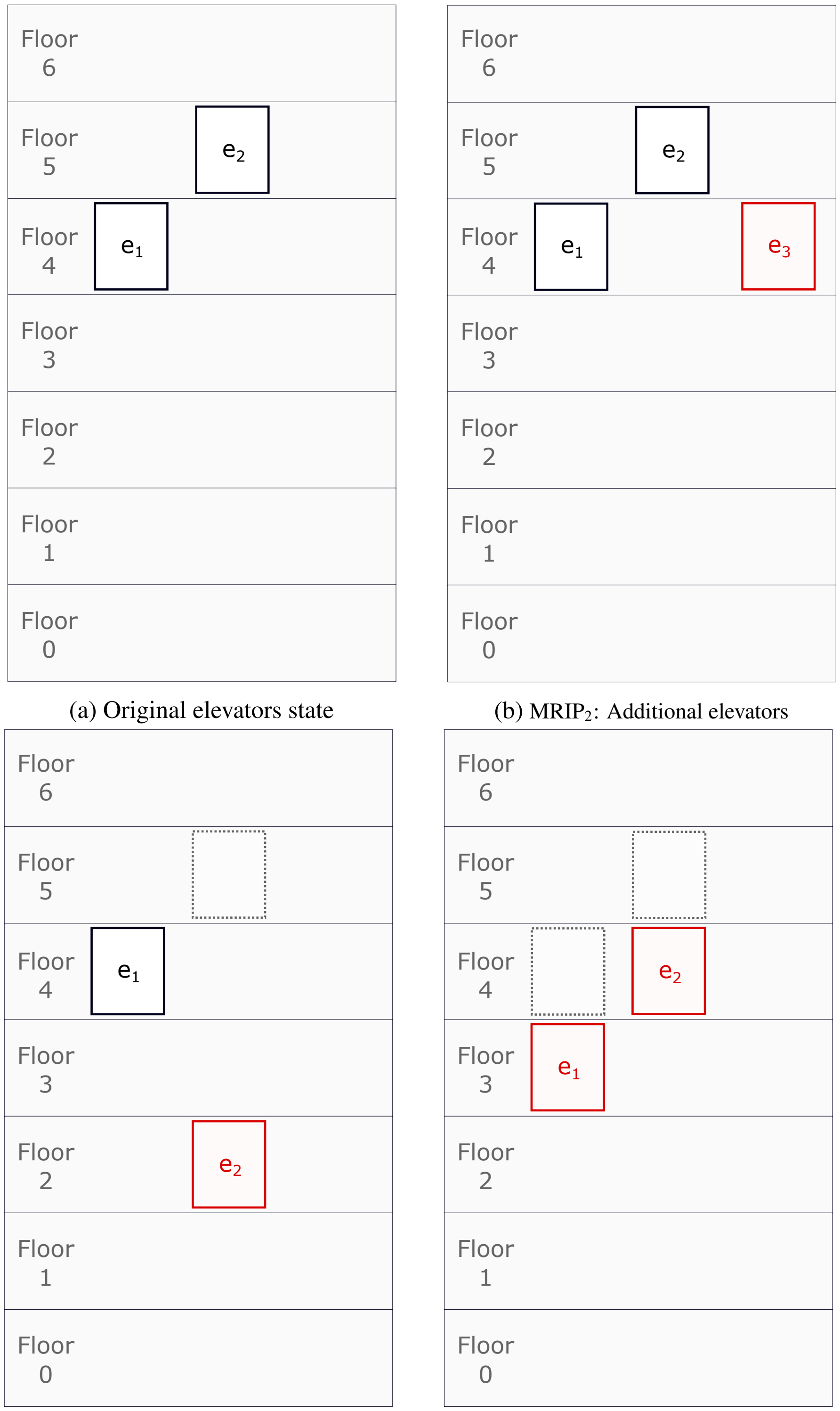
In order to describe our MRs, we define the operation $serve(E, S, C)$ for elevator installations, where E is the list of floor numbers where the elevators are positioned initially, S is the nominal speed of the elevators, and C is a set of passenger calls $c \in C$, each of which will be encoded as (c_t, c_s, c_d), representing an arrival time (c_t), a source floor (c_s), and a destination floor (c_d). Figure 7.1a shows an example scenario of a 6-story building with two elevators in floors 4 and 5, which we encode as $E = \{4, 5\}$. For simplicity, we assume that the speed parameter S applies to all the elevators equally, i.e., all the elevators are identical.

MRIP1: Additional calls. This MRIPs represents MRs where the follow-up test inputs are constructed by adding one or more passenger calls to the source input. When this happens, the performance of the system is generally expected to be the same or worse, since there is extra work for the elevators to perform. For instance, the total distance (TD) traversed by the elevators should increase or remain the same in the follow-up test case, since the elevators need to attend to additional passengers. This MR can be expressed as follows:

$$TD(serve(E, S, C_s)) \lesssim TD(serve(E, S, C_f)) \tag{7.1}$$

where $C_f = C_s \cup c$.

For instance, suppose a source test case consisting of the initial elevator positions from Figure 7.1a, $E = \{4, 5\}$, nominal speed $S = 1$, and the set of passenger calls $C_s = \{(1, 2, 3)\}$, representing a single call at $t = 1$ from floor 2 to floor 3. A follow-



(a) Original elevators state

(b) MRIP$_2$: Additional elevators

(c) MRIP$_4$: Initial position change

(d) MRIP$_5$: Shift Floors

Figure 7.1: Elevator system scenarios.

up test case can be created by adding a new call at $t = 2$ from floor 5 to floor 3: $C_f = \{(1,2,3),(2,5,3)\}$. In this case, the TD should increase or remain the same:

$$TD(serve(\{4,5\},1,\{(1,2,3)\})) \lesssim TD(serve(\{4,5\},1,\{(1,2,3),(2,5,3)\})) \quad (7.2)$$

MRIP2: Additional elevators. This MRIPs describes the relations where the follow-up test input is generated by adding new elevators to the source input. In this case, the performance of the system should generally be expected to improve, since the elevator control system has more resources to work with. For example, the following MR is an instance of this pattern, where adding one ore more elevators is expected to decrease the Average Waiting Time (AWT) for the passengers:

$$AWT(serve(E_s,P,C)) \gtrsim AWT(serve(E_f,P,C)) \quad (7.3)$$

where $E_f \supset E_s$.

Figure 7.1b shows a possible follow-up of this MR for the scenario from Figure 7.1a. An additional elevator e_3 is added at *floor 4* in the follow-up test case, resulting in $E_f = \{4,5,4\}$. Consider the arbitrary nominal speed $S = 1$ and passenger calls $C = \{(1,2,3)\}$. In this scenario, the AWT should decrease or remain the same after adding the new elevator:

$$AWT(serve(\{4,5\},1,\{(1,2,3)\})) \gtrsim AWT(serve(\{4,5,4\},1,\{(1,2,3)\})) \quad (7.4)$$

MRIP3: Faster elevators. This pattern groups the relations where the nominal speed of the elevators is increased for the follow-up test case ($S_f > S_s$). As an example, the AWT of the follow-up test case is expected to improve due to the elevators being able to attend calls faster, hence the following MR:

$$AWT(serve(E,S_s,C)) \gtrsim AWT(serve(E,S_f,C)) \quad (7.5)$$

where $S_f > S_s$.

For example, suppose the initial elevator positions from Figure 7.1a, the set of passenger calls $C_s = \{(1,2,3)\}$, and the nominal speed $S_s = 1$. A follow-up test case can be created by doubling the nominal speed of the elevators ($S_f = 2$). In this scenario, we should expect the AWT of the follow-up test case to be lower, or at worst similar to the one obtained in the source test case:

$$AWT(serve(\{4,5\},1,\{(1,2,3)\})) \gtrsim AWT(serve(\{4,5\},2,\{(1,2,3)\})) \quad (7.6)$$

MRIP4: Initial position change. This pattern represents the MRs where the initial positions of the elevators E_s are randomly changed to any other positions E_f, but without changing the number of available elevators ($|E_f| = |E_s|$). In this case, most of the metrics such as the AWT should remain similar as long as the test case is long enough, since the initial positions of the elevators should have an almost negligible effect in longer scenarios:

$$AWT(serve(E_s, S, C)) \simeq AWT(serve(E_f, S, C)) \tag{7.7}$$

where $E_f \neq E_s$ and $|E_f| = |E_s|$.

For example, suppose the initial elevator positions from Figure 7.1a are changed to those from Figure 7.1c ($E_f = \{4, 2\}$), and we have the set of passenger calls $C_s = \{(1, 2, 3), (2, 5, 3)\}$, and the nominal speed $S = 1$. We would expect the AWT of both test cases to be similar:

$$AWT(serve(\{4,5\}, 1, \{(1,2,3),(2,5,3)\})) \simeq AWT(serve(\{4,2\}, 1, \{(1,2,3),(2,5,3)\})) \tag{7.8}$$

MRIP5: Shift floors. This MRIPs represents MRs where all the positions for both the elevators (E) and passenger calls (C) are offset by the same amount of floors. In this case, all the metrics, such as AWT, would intuitively remain very similar, since both scenarios are identical in terms of the relative positions between elevators and passengers. Here, we are assuming that all the floors and distances between them are identical. We can define the MR based on AWT as follows:

$$AWT(serve(E_s, S, C_s)) \simeq AWT(serve(E_f, S, C_f)) \tag{7.9}$$

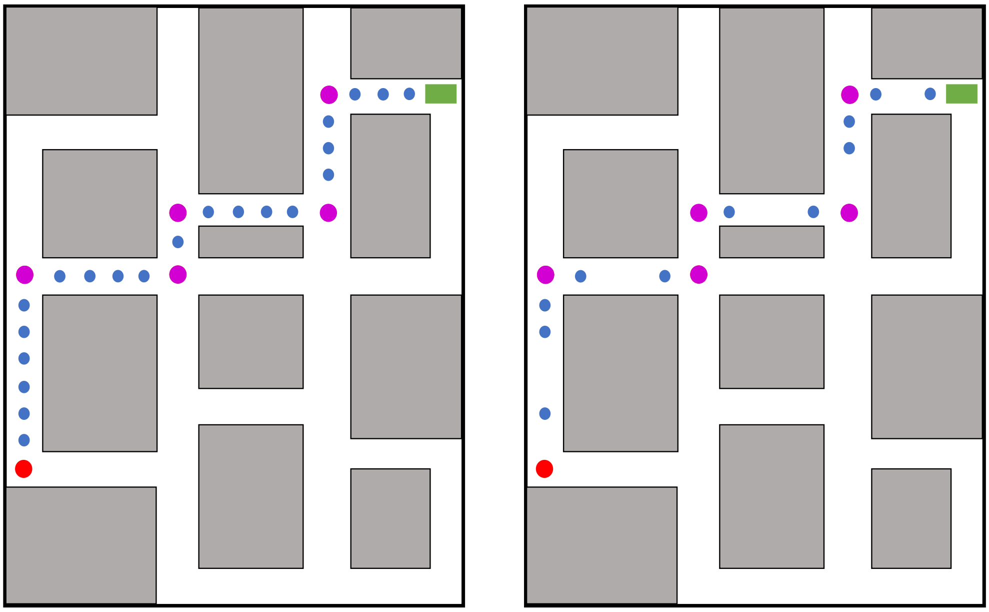
where E_f and C_f are generated by offsetting the positions of E_s and C_s by a constant number of floors.

For instance, consider the source test case from Figure 7.1a, where $E_s = \{4, 5\}$, $S = 1$, and $C_s = \{(1, 2, 3)\}$. A follow-up test case can be generated by offseting all the positions to be one floor below, i.e., $E_f = \{3, 4\}$ and $C_f = \{(1, 1, 2)\}$. In this case, the AWT should remain the same in both scenarios:

$$AWT(serve(\{4,5\}, 1, \{(1,2,3)\})) \simeq AWT(serve(\{3,4\}, 1, \{(1,1,2)\})) \tag{7.10}$$

7.2.2 Autonomous Driving System

ADSs are capable of planning the route of a vehicle and driving through it without any human intervention. These vehicles (henceforth referred to as *autonomous vehicles*)



(a) Original elevators state

(b) MRIP$_4$: Fewer guidance points

Figure 7.2: ADS scenarios.

may be cars, ships, drones or submarines, for example, but in this work we employ a car with a waypoint-based navigation system. The autonomous car can specifically: (1) Plan an efficient route from the source to the destination points, (2) traverse the route while keeping track of its current position, and (3) identify and avoid obstacles in its path.

In order to formulate MRs for this system, we represent its functionality as the operation $move(P, p_A, p_B, S, O)$, where P is a set of guidance points known by the ADS, p_A is the origin point (the vehicle's initial position), p_B is the destination point, S is the vehicle's nominal speed, and O is the set of obstacles in the environment, which should be avoided by the vehicle. Note that both the origin and destination points must be part of the set of guidance points from the map ($p_A \in P$ and $p_B \in P$). Figure 7.2a shows an example scenario where the vehicle (green) must traverse several guidance points (blue and purple) to reach the destination point (red). We use an autonomous car to illustrate scenarios for the proposed MRs because this is the type of vehicle we use in our empirical evaluation, but the MRs described in this section are also applicable to drones, boats, and other vehicles, as long as their functionality can be described with the $move$ operation we have defined.

MRIP1: Faster vehicles. This pattern represents MRs where the vehicle's nominal speed is increased in the follow-up test case ($S_f > S_s$). In terms of Time To Destination (TTD), we would expect a similar or lower time, since the vehicle can traverse its route faster as long as it can accelerate to its nominal speed.

The expected performance should be the same or better in terms of travel time, since the vehicle can traverse its route faster as long as it can accelerate to its nominal speed. Thus, we define the following MR:

$$TTD(move(P, p_A, p_B, S_s, O)) \gtrsim TTD(move(P, p_A, p_B, S_f, O)) \tag{7.11}$$

where $S_f > S_s$.

For instance, consider a scenario for a self-driving car where the route comprises the waypoints $P = \{w1, w2, w3\}$, the starting point is $p_A = w1$, the goal is $p_B = w3$, and there are no obstacles ($O = \{\}$). If the nominal speed $S_s = 60$ is increased for the follow-up test case ($S_f = 80$), the time to destination should decrease:

$$TTD(move(\{w1, w2, w3\}, w1, w3, 60, \{\})) \gtrsim TTD(move(\{w1, w2, w3\}, w1, w3, 80, \{\})) \tag{7.12}$$

MRIP2: Additional obstacles. This MRIP groups the MRs where follow-up test cases are created by adding obstacles to the environment where the vehicle operates ($O_f \supset O_s$). In this case, the TTD should be expected to increase, since the vehicle must overcome the new obstructions in its path by taking otherwise unnecessary actions. Obstacles may include static or dynamic objects, such as a cone in the middle of the path or another vehicle driving through the same road. For TTD, we can define following MR:

$$TTD(move(P, p_A, p_B, S, O_s)) \lesssim TTD(move(P, p_A, p_B, S, O_f))) \tag{7.13}$$

where $O_f \supset O_s$.

For the autonomous car example, consider a scenario where an obstacle, such as a cone in the middle of the road is introduced as an obstacle, resulting in the car having to steer to avoid it. Consider the inputs $P = \{w1, w2, w3\}$, $p_A = w1$, $p_B = w3$, $S = 60$, with $O_s = \{\}$ and $O_f = \{cone\}$. The alternative trajectory should be less optimal than the original one, resulting in an increase on the TTD:

$$TTD(move(\{w1, w2, w3\}, w1, w3, 60, \{\})) \lesssim TTD(move(\{w1, w2, w3\}, w1, w3, 60, \{cone\})) \tag{7.14}$$

MRIP3: Reversed path. This pattern represent MRs where the path P_s is reversed in P_f. Intuitively, this should result in the source and follow-up test executions being very similar in most cases, since the same path is traversed in both cases, only in opposite directions. This MRIP has already been used multiple times in previous publications [CHTZ04, SFSRC16], but here we use it to instantiate MRs based on performance metrics. For example, the corresponding MR for TTD would be:

$$TTD(move(P_s, p_A, p_B, S, O)) \simeq TTD(move(P_f, p_B, p_A, S, O))) \tag{7.15}$$

where P_f is obtained by reversing the order of the waypoints in P_s.

As an example, consider the inputs $P_s = \{w1, w2, w3\}$, $p_A = w1$, $p_B = w3$, $S = 60$, and $O = \{\}$. The corresponding relation after reversing the waypoints to $P_f = \{w3, w2, w1\}$ would be:

$$TTD(move(\{w1, w2, w3\}, w1, w3, 60, \{\})) \simeq TTD(move(\{w3, w2, w1\}, w3, w1, 60, \{\})) \tag{7.16}$$

MRIP4: Fewer guidance points. This MRIPs corresponds with the MRs where some of the guidance waypoints from the path of the vehicle are removed in the follow-up test case ($P_f \subset P_s$). Here, we assume that there are enough waypoints such that omitting some of them will still allow the vehicle to follow along a very similar path, as shown in Figure 7.2b. With this transformation the traversed path will be very similar, so the TTD should also remain similar:

$$TTD(move(P_s, p_A, p_B, S, O)) \simeq TTD(move(P_f, p_A, p_B, S, O))) \tag{7.17}$$

where $P_f \subset P_s$.

Consider the same inputs as the other examples $P_s = \{w1, w2, w3\}$, $p_A = w1$, $p_B = w3$, $S = 60$, and $O = \{\}$. The follow-up path could be generated by omitting the intermediate waypoint $w2$, resulting in $P_f = \{w1, w3\}$. Then, the relation would be:

$$TTD(move(\{w1, w2, w3\}, w1, w3, 60, \{\})) \simeq TTD(move(\{w1, w3\}, w3, w1, 60, \{\})) \tag{7.18}$$

7.3 Evaluation

This section describes two empirical experiments that aim to assess the cost-effectiveness of applying the PV pattern for testing CPSs. We specifically aim to answer the following Research Questions (RQs):

- **RQ1**. *Do the MRs trigger false positives, and if so, what is their cause?* False positives are likely to emerge due to the unpredictable behaviour of our Systems Under Test (SUTs), especially since CPSs encompass physical components and an environment which is often uncertain to some extent. We aim to investigate to what extent false positives can happen in practice.

- **RQ2**. *Is the PV pattern effective in revealing failures in CPSs?* We aim to study the effectiveness of the proposed MRP in terms of detecting failures in CPSs. Due to the lack of specified oracles for our experimental subjects, we will employ automated regression test oracles as a baseline for the results obtained by our MRs.

- **RQ3**. *Do particular MRIPs or performance metrics perform significantly better than others?* We aim to compare the performance of different MRIPs and performance metrics in order to determine whether some of them are subsumed or complemented by others. Due to the high cost of MT and CPS testing, it is particularly interesting to know if certain MRs can be skipped without sacrificing effectiveness.

Table 7.1: Main characteristics of the experimental case studies [AVS$^+$22a]

SUT	**Language**	**Tests (source + follow-up)**	**Execution time**	**Mutants**	**MRs**	**Metrics**
Elevator	C	1340 (140 + 1200)	∼60 hours	89	9	3
ADS	Simulink	1300 (100 + 1200)	∼8 hours	20	12	2

The main features of the two case studies employed in this empirical evaluation are summarized in Table 7.1.

7.3.1 Evaluation metrics

This section presents the key definitions and metrics used for describing the experimental results of both experiments.

We define a *metamorphic test* as a check of the output relation of an MR, which is preceded by the execution of a pair of source and follow-up test cases that satisfy the corresponding input relation. If the output relation is violated, the metamorphic test is said to have failed, indicating a *test failure*.

Since our experimental setup involves mutation testing, the correctness of the test executions is known a priori. We refer to a *test failure* over a pair of test executions that is supposed to be correct as a *False Positive (FP)*.

We employ three different evaluation metrics to determine the effectiveness of our approach. First, we report the *FPs*, which is the ratio of test failures on the original system executions. FPs may result in unnecessary debugging efforts, so the lower the ratio of FPs the better. Second, we report the *Mutation Score (MS)*, which is defined as the ratio of mutants killed by a MR. Specifically, we consider a mutant as “detected" or “killed" when one or more of the metamorphic tests on it resulted in a failure, and the corresponding metamorphic test on the original system did not. The higher the mutation score, the better, since more seeded faults are detected. Third, we also measure the *Failure Detection Ratio (FDR)*, which is the ratio of metamorphic tests on mutants that resulted in a test failure. A higher FDR is better, since it indicates that more faulty behaviours are identified. In general, the MS represents the *diversity* of faults detected, whereas the FDR represents their *quantity*.

7.3.2 Experiment 1: Elevator Control System

The first experiment is based on the industrial elevator control system developed by Orona [Oro22], which was introduced in Section 4.4.1. This case study also inspired the example MRs presented in Section 7.2.1, and the MRs we have implemented are more refined versions of some of the ones presented in that section. In what follows, we describe the implemented MRs, the experimental setup, and the results of the experiment.

Metamorphic Relations

In this section, we define the MRs we have implemented for the elevation control system based on the QoS metrics described in Section 4.4.1, namely Average Waiting Time (AWT), Total Distance (TD), and Total Movements (TM).

The following MRs are defined assuming the elevator control system always provides an optimal assignment. Unfortunately, a real elevator system does not have the information needed to yield an optimal response, since this would require knowing the passenger calls and destinations *beforehand*. In practice, this means that false positives could arise for most of these MRs, since there is no objective way of distinguishing between suboptimal behaviour which can be tolerated and actual faulty behaviour. To mitigate this, as explained in Section 7.2, we define approximate relations ($\simeq, \gtrsim, \lesssim$) instead of strict ones ($=, \geq, \leq$). In practice, these are implemented using tolerance thresholds for some of the MRs, meaning that a only violations exceeding a certain value will be considered as test failures. This way, most of the tolerable suboptimal behaviours will not result in a test failure. The specific threshold values used for each MR are detailed in Section 7.3.2.

For simplicity, we still use the same notation introduced in Section 4.4.1 for describing the system functionality, where $serve(E, S, C)$ denotes an execution of the dispatcher, E is a set of floors indicating the positions of the elevators, S is the nominal speed of the elevators, and C is a list of passenger calls.

MRIP$_1$: Additional calls. We propose several MRs based on the MRIP from Equation 7.1 in Section 7.2.1, where the passenger calls from the follow-up test input are a superset of those in the source test inputs $C_f = C_s \cup c'$. More specifically, we append a single passenger call (c') as the last call (arrival time at the same time of the last call in C_s or later). With this MRIP, the TD traversed by the elevators should increase, since there is one more passenger call to attend to. Nevertheless, we found that it is possible to define a tighter —and therefore more likely to reveal failures [SFSRC16]— MR by making an estimation of the worst case distance required to be traversed for the additional call, measured as the sum of the largest possible distance to the source floor and the distance between the source and the destination floors. Thus, we define the following MR:

$$TD(serve(E, S, C_f)) \lesssim TD(serve(E, S, C_s)) + TD_w(c') \qquad (\texttt{MR1}_{\texttt{TD}})$$

where $TD_w(c')$ is the worst case distance that an elevator will have to traverse for serving c', calculated as: $TD_w(c) = max(c_s - 1, FLOORS - c_s) + |c_s - c_d|$, where $max(c_s - 1, FLOORS - c_s)$ is the longest possible distance that may need to be traversed to reach the source floor c_s, and $|c_s - c_d|$ is the distance from the source floor to the destination floor of the passenger.

A similar relation is defined based on the expected impact on the AWT:

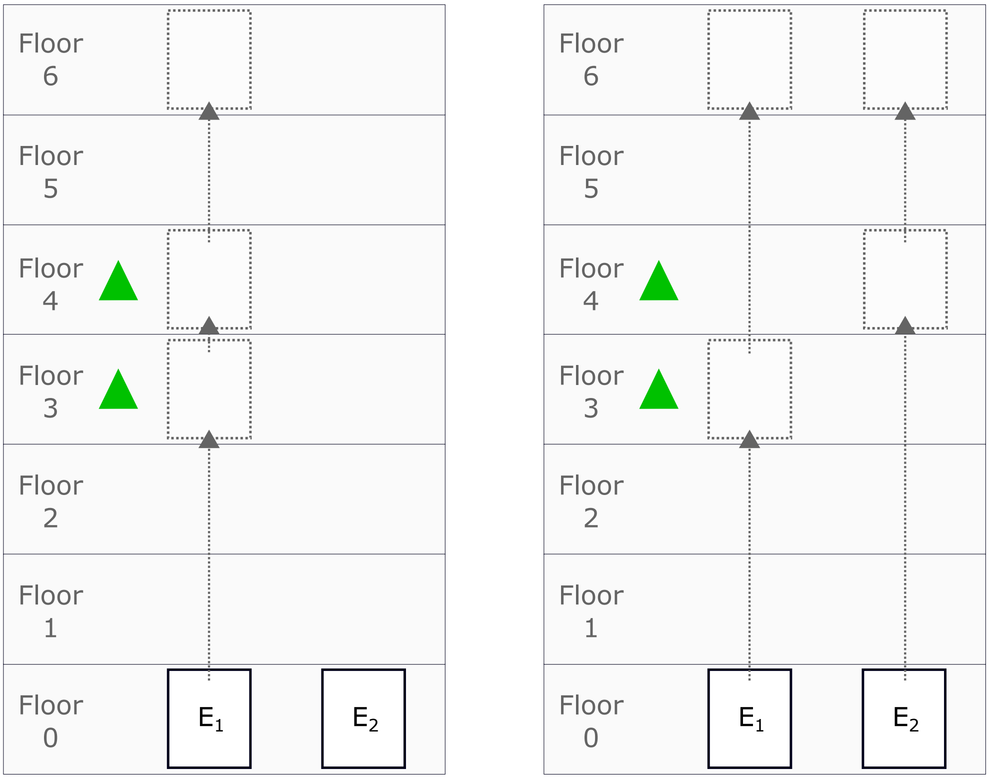
$$AWT(serve(E, S, C_f)) \lesssim AWT(serve(E, S, C_s)) + WT_w(c') \qquad (\texttt{MR1}_{\texttt{AWT}})$$

where $WT_w(c')$ is the estimated worst case waiting time for c', calculated as $T(max(c_s - 1, FLOORS - c_s))$, where $max(c_s - 1, FLOORS - c_s))$ is the longest possible waiting distance that was also used for $TD_w(c')$, and $T(distance)$ is a formula which calculates the time in seconds that it takes an elevator to traverse the given distance considering its speed, acceleration and jerk.

Finally, the TM should increase or remain the same for attending the extra call:

$$TM(serve(E, S, C_f)) \gtrsim TM(serve(E, S, C_s)) \qquad (\texttt{MR1}_{\texttt{TM}})$$

MRIP$_2$: Additional elevators. Based on the MRIP from Equation 7.3 in Section 7.2.1, we define MRs where one or more additional elevators are enabled for the follow-up test case. Formally, this input relation can be defined as $E_f = E_s \cup E'$, where E' is a nonempty set of new elevators. This provides the elevator control



(a) Solution for best TD

(b) Solution for best AWT

Figure 7.3: Conflicting objectives in elevator control systems.

system additional workers to perform its tasks (fulfilling passenger requests), so the performance of the installation should generally improve. Since the main objective of the elevator dispatching algorithm we are testing is optimizing for the lowest possible AWT, we can expect this metric to be reduced, or at least not increase:

$$AWT(serve(E_f, S, C)) \lesssim AWT(serve(E_s, S, C)) \tag{$\texttt{MR2}_{\texttt{AWT}}$}$$

Conversely, the TD is likely to increase if more elevators are moving in parallel, since this metric conflicts with improving the AWT by attending calls in parallel with multiple elevators. Figure 7.3 shows two conflicting solutions for the same scenario, where Figure 7.3a obtains the best TD, but Figure 7.3b obtains a better AWT (TD is doubled, but the passenger in floor 4 has a lower waiting time because it is attended directly). Our SUT in this case would expected to favour the solution with a better AWT. This is reflected in the following MR:

$$TD(serve(E_f, S, C)) \lesssim TD(serve(E_s, S, C)) \cdot (1 + |E_f| - |E_s|) \tag{$\texttt{MR2}_{\texttt{TD}}$}$$

where $|E_f| - |E_s|$ is the upper bound of the traversed distance based on the number of additional elevators. For instance, if we add 2 more elevators, the TD could be increased by up to 200% in the worst case.

For the same reason as TD, the TM can also be expected to increase if new elevators are added:

$$TM(serve(E_f, S, C)) \lesssim TM(serve(E_s, S, C)) \cdot (1 + \frac{|E_f| - |E_s|}{2}) \qquad (\texttt{MR2}_{\texttt{TM}})$$

where $\frac{|E_f| - |E_s|}{2}$ is the upper bound of the traversed distance based on the number of additional elevators. In this case, TM is only expected to increase by up to 50% for each extra elevator because the number of movements to move the passengers to their destination is the same either way, and only the movements to attend the calls may increase. For instance, if we compare the scenario in Figure 7.3a and the one in Figure 7.3b, they perform 3 and 4 movements respectively.

MRIP$_3$: Faster elevators. Following the MRIP from Equation 7.5 in Section 7.2.1, we define MRs where the speed of the elevators is increased in the follow-up test case. Formally, $S_f > S_s$. Note that S affects all the elevators from the installation equally. This change should generally reduce the AWT, since the faster elevators should be able to attend the calls faster, hence the following relation:

$$AWT(serve(E, S_f, C)) \lesssim AWT(serve(E, S_s, C)) \qquad (\texttt{MR3}_{\texttt{AWT}})$$

As for the TD, it can be expected to increase instead. The reason for this is that the slower elevators will cause more passenger calls to accumulate, resulting in more passengers travelling together on average. As a result, the distance and number of movements that the slower elevators will have to perform will generally be reduced. For instance, consider the scenario depicted in Figure 7.3a, where the elevator is able to pick up the passengers coming from floor 3 and 4 and have them travel together to floor 6. If the passenger from floor 4 calls for the elevator 20 seconds before the passenger from floor 3 does, an elevator which can reach floor 4 within 20 seconds would not pick the passenger from floor 3 on their way, and would instead travel to the first passenger's destination, and then make another trip attend the call from floor 3. Thus, we define the following MR:

$$TD(serve(E, P_f, C)) \gtrsim TD(serve(E, P_s, C)) \qquad (\texttt{MR3}_{\texttt{TD}})$$

For the same reason, TM is also expected to increase:

$$TM(serve(E, P_f, C)) \gtrsim TM(serve(E, P_s, C)) \qquad (\texttt{MR3}_{\texttt{TM}})$$

Experimental Setup

Test case template. The test cases used for our experiments are based on a template project from a real multi-elevator installation with 10 floors and 6 elevators. For simplicity, the 10 floors have the same height, and all the elevators use identical configurations. Thus, E can be a set of size 2 to 6 (at least 2 elevators must be enabled) of positions (floors 1 through 10), S corresponds with the real installation's configuration unless the MRIP changes it, and C is a nonempty list of passenger calls, each of which must have different source and destination floors (1 through 10).

Source test cases. The source test cases were randomly generated based on the template described above. Each test case lasts an average of 3 minutes. For each generated test case, we selected a random number of elevators (between 2 and 6), a random initial floor for each elevator (between 1 and 10), and a random passenger list generated by uniformly distributing the calls across a fixed time period, as well as randomly selected source and destination floors. In total, 140 random source test cases were generated.

Follow-up test cases. The follow-up test cases for MRIP1 (additional calls) were generated by appending a single additional call to the end of the passengers list. The arrival time of the new calls was set to a random time between the last call and 30 seconds after it, such that: $last(C_s) \leq last(C_f) \leq last(C_s) + 30$. For MRIP2 (additional elevators), follow-up test cases were generated by randomly selecting a number of elevators $|E_f| > |E_s|$, but also respecting that $|E_f| \leq 6$. The additional elevators were given random initial positions, and the positions of the ones that were in E_s remained unchanged. For MRIP3 (faster elevators), the speed of the elevators was scaled: $S_f = S_s \cdot m$, where $m \in \{2, 3, 4\}$. In total, 1200 pairs of source and follow-up test cases were generated: 420 for MRIP1, 360 for MRIP2, and 420 for MRIP3.

Mutants. We employed mutation testing in order to assess the effectiveness of our approach. Just et al. found that seeding artificial faults is a valid way of estimating the ability for finding real faults [JJI$^+$14]. For this experiment, we generated 89 mutants of Orona's elevator dispatching algorithm by seeding faults using traditional mutation operators [ADH$^+$89]. The faults were seeded manually in a uniform manner by a domain expert. The 140 source test cases and 1200 follow-up test cases were executed in the original SUT, as well as the 89 mutants, resulting in a total of $(140 + 1200) \times (1 + 89) = 120,600$ individual test executions. The test execution outputs for all the SUTs were compared with each other in order to ensure that none of them were semantically equivalent.

Thresholds. As previously mentioned, we defined tolerance thresholds for some of the MRs in order to mitigate possible FPs. After some preliminary tests, we defined a threshold of 30% for MR3$_{\text{TD}}$ and MR3$_{\text{TM}}$. For instance, MR3$_{\text{TD}}$ was implemented as: $TD(serve(E, P_f, C)) \geq TD(serve(E, P_s, C)) \cdot 0.7$. The remaining MRs were implemented by evaluating them strictly, just replacing the approximate operators for their strict equivalent (e.g. $\lesssim$ becomes $\leq$). For example, MR3$_{\text{AWT}}$ was evaluated as $AWT(serve(E, P_f, C)) \leq AWT(serve(E, P_s, C))$.

Baseline

Table 7.2: Main characteristics of the test cases used by the baseline approach [AVS+22a]

Test case	Up Calls	Down Calls	Killed Mutants	Duration (h:min)
real1	2756	1711	18	8:30
real2	3086	2366	18	9:10
real3	3438	3117	18	11:45
real4	3508	3050	21	13:35
theoretical1	3994	3377	20	12:55
theoretical2	3950	3379	18	12:55
theoretical3	3983	3379	26	12:55
theoretical4	3989	3402	18	12:55
theoretical5	3989	3387	18	12:55
theoretical6	3964	3384	19	12:55
theoretical7	3977	3386	21	12:55
theoretical8	3919	3433	21	12:55
theoretical9	3976	3354	18	12:55
theoretical10	3945	3407	20	12:55

The current practice for verifying elevator dispatching algorithm versions at Orona at the Software-in-the-Loop (SiL) level includes the use of an automated regression test oracle [AAI+21]. These oracles evaluate the QoS measures obtained by the SUT over time by comparing it with the measures obtained by a golden implementation, which is usually a previous version of the SUT. The regression oracles used at Orona specifically check three different properties of the AWT: (1) No major regressions must occur within a single 5 minute period, (2) no accumulated regressions over multiple 5 minute periods, and (3) no significant regression in the global AWT. These oracles are typically applied over a test suite comprised by 14 full-day traffic profiles, 4 of which are obtained from real installations, while the remaining 10 are based on theoretical traffic profiles. Table 7.2 summarizes the key characteristics of these test cases.

In order to provide a fair comparison between our MRs and Orona's current automated testing approach, we perform mutation testing with the same 89 mutants, but we have this baseline use the test cases from Table 7.2 because that is the test suite that is used in reality. Out of the 89 mutants, 8 were unable to finish the simulations due to getting stuck (some passengers were never attended), and we will consider killed for our evaluation because an implicit timeout can easily detect the failure.

The total simulation time of all of the scenarios is 10,330 minutes, or approximately 7 days. However, the regression oracles require the execution for two systems, resulting in double the execution costs. Therefore, the total execution time employed by this approach is 20,660 minutes, or approximately 14 days.

Experimental Results

Table 7.3: Evaluation results on the elevator dispatcher (MS: Mutation Score, FDR: Fault Detection Ratio, FP: False Positives) [AVS+22a]

MRIP	MR	MS (%)			FDR (%)	FP (%)
MRIP1	$MR1_{AWT}$	29.21	85.39	88.76	0.19	0.00
	$MR1_{TD}$	65.17			0.88	0.24
	$MR1_{TM}$	75.28			0.98	0.24
MRIP2	$MR2_{AWT}$	42.70	42.70		2.27	0.00
	$MR2_{TD}$	13.48			0.08	0.00
	$MR2_{TM}$	5.62			0.03	0.00
MRIP3	$MR3_{AWT}$	31.46	44.94		0.55	0.24
	$MR3_{TD}$	33.71			0.24	0.00
	$MR3_{TM}$	6.74			0.06	0.00

Table 7.3 shows the experimental results obtained from the elevator control system experiments, including the FP, MS, and FDR metrics. In what follows, we discuss the results.

The experiments resulted in a single false positive (FP ratio of 0.24%) for three of the MRs, namely: $MR1_{TD}$, $MR1_{TM}$ and $MR3_{AWT}$. These false positives were manually checked with the help of domain experts in order to determine their cause. After some investigation, we found that one of the cases was due to the Elevate simulator sending information to the elevator dispatching algorithm in a way that Orona's controllers never do, which caused the algorithm to misbehave. This discrepancy is known by the engineers at Orona, and is recognized and ignored when observed due to the cost of changing the dispatcher or the controllers to fix it not being justifiable. Another case was caused by an elevator skipping a passenger call in a scenario where stopping for the passenger seemed to be a more obvious choice, which affected the

QoS significantly because it was a rather short test case. This was caused by an heuristic of the algorithm which provides a better QoS on average, even though it can cause suboptimal behaviour in specific instances. Reporting a failing condition could be considered acceptable in both of these cases, since both issues were identified and fixes were attempted at some point, although leaving the algorithm unchanged was found to be the preferable choice in the end. on the other hand, the false positive from $\mathrm{MR3}_{\mathrm{AWT}}$ was found to be simply a statistical anomaly, where making the elevators faster caused the scenario to diverge in an unfavourable way, which is a possible but unlikely occurrence.

As for the overall effectiveness, the combined MS for our MRs was 88.76% (79 out of 89 mutants killed). As for the FDR, the combined result was 0.5% (1,593 out of 320,400 metamorphic test failures). Note that we employ 89 mutants, with 1200 test pairs and 3 different MRs which can be checked for each of the metamorphic test pairs, so the number of metamorphic tests is $89 \times 1200 \times 3 = 320,400$.

As shown in Table 7.3, the MS of individual MRs ranged from 5.62% to 75.28%, whereas the FDR ranged from 0.03% to 2.27%, which implies that their effectiveness varies greatly. A combination of $\mathrm{MR1}_{\mathrm{TD}}$ and $\mathrm{MR1}_{\mathrm{TM}}$ would seem to yield the best results in terms of cost-effectiveness, since they provide a MS close to the global aggregate and an above-average FDR. $\mathrm{MR2}_{\mathrm{AWT}}$ seems to be an outlier in terms of FDR, and might also be cost-effective to use. These three MRs combined obtain an aggregate MS of 87.64%, which is only one less mutant killed than with the aggregate of all the MRs.

Comparing the results with the baseline presented in Section 7.3.2, which employs regression oracles, the MRs killed 32 additional mutants. In terms of MS, the results are 52.8% for the baseline and 88.76% for our approach, which is a very significant improvement in effectiveness. On the other hand, since both approaches employ test cases with very different features, there is no meaningful way to compare the FDRs. As for the cost, which we measure in terms of execution time, the total cost of running all of our metamorphic tests was 3678.62 minutes, or approximately 2 days and a half. As for the baseline, the total cost of the test suite was 10,330 minutes, or approximately 7 days, although the actual cost of the baseline could be considered to be twice as much if the reference implementation needs to be run as well. In any case, our approach reduces the cost of testing by over a factor of 3. Nevertheless, our MRs resulted in 3 FPs, which may lead to some unnecessary efforts from the domain experts, whereas the baseline approach is considered to not have FPs due to the way it is formulated (performance regressions are assumed to never be false positives).

Next, we discuss the MSs obtained by the MRs grouped by MRIP or QoS metric

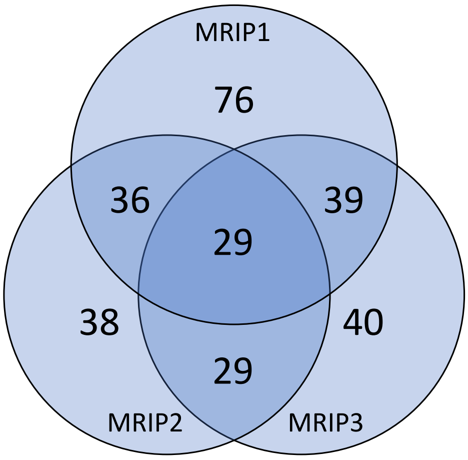


Figure 7.4: Mutants killed per MRIP (out of 89) [AVS+22a]

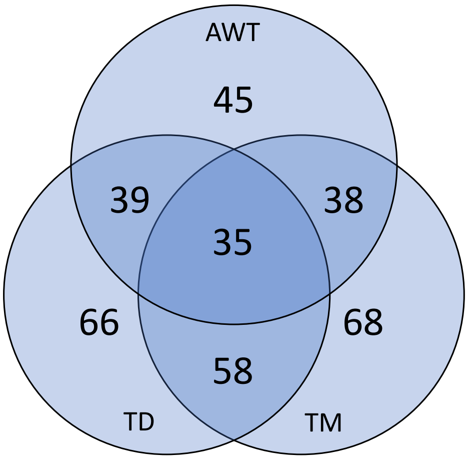


Figure 7.5: Mutants killed per QoS metric (out of 89) [AVS+22a]

in order to identify how they compliment or subsume each other. Figure 7.4 shows the number of mutants killed by the MRs grouped by MRIP, whereas Figure 7.5 shows the MSs grouped by QoS metric. Looking at Figure 7.4, MRIP1 (additional calls) is the most effective pattern in terms of MS by a large margin, and only three additional mutants that can be killed by other MRIPs. Nevertheless, the fact that each of its two best MRs yielded a false positive should be considered. As for Figure 7.5, AWT is shown to be the least effective metric, even though this QoS metric is the main optimization objective for our SUT. Despite this, AWT-based MRs still kill 6 mutants missed with TD and 7 mutants missed with TM. It is also worth noting that the AWT-based MRs obtained the best FDRs with both MRIP2 and MRIP3.

7.3.3 Experiment 2: Autonomous Driving System

The second experiment is based on the open-source Autonomous Driving System (ADS) published by MathWorks [Mat], introduced in Section 4.4.2. This case study inspired the example MRs presented in Section 7.2.2, and the MRs we have implemented are a superset of the ones presented in that section. next, we describe the implemented MRs, the experimental setup, and the results of this experiment.

Metamorphic Relations

In this section, we define the MRs we have implemented for the ADS. We define these MRs based on the QoS metrics described in Section 4.4.2, namely Time To Destination (TTD) and Total Trajectory Offset (TTO).

Just like in the previous experiment, we define approximate relations ($\simeq, \gtrsim, \lesssim$) implemented with tolerance thresholds, and later detail the specific threshold values we used in Section 7.3.3.

We still use the same notation introduced in Section 4.4.2 for describing the system functionality, where $move(P, p_A, p_B, S, O)$ denotes an execution of the ADS, P is a set of waypoints in the map, p_A and p_B are the origin and destination of the vehicle, and S is the nominal speed of the vehicle, and O is a set of obstacles (other vehicles).

MRIP$_1$: Faster vehicles. We propose several MRs based on the MRIP from Equation 7.11 in Section 7.2.2, where the follow-up test input is created by increasing the original nominal speed ($S_f > S_s$). With this MRIP, the TTD can be expected to be reduced in the follow-up test case, since a faster vehicle should be able to reach its destination in less time:

$$TTD(move(P, p_A, p_B, S_f, O)) \lesssim TTD(move(P, p_A, p_B, S_s, O)) \qquad (\texttt{MR1}_{\texttt{TTD}})$$

where $S_f > S_s$.

Conversely, the TTO can be expected to increase, since steering precisely becomes harder the faster the vehicle is driving. Thus, we define the following MR:

$$TTO(move(P, p_A, p_B, S_f, O)) \gtrsim TTO(move(P, p_A, p_B, S_s, O)) \qquad (\texttt{MR1}_{\texttt{TTO}})$$

MRIP$_2$: Additional obstacles. Based on the MRIP from Equation 7.13 in Section 7.2.2, we define MRs where an extra obstacle is placed in the vehicle's trajectory ($O_f \supset O_s$). Due to the interference from the additional obstacle, the TTD is expected to increase, hence the following MR:

$$TTD(move(P, p_A, p_B, S, O_f)) \gtrsim TTD(move(P, p_A, p_B, S, O_s)) \qquad (\texttt{MR2}_{\texttt{TTD}})$$

where $O_f \supset O_s$.

On the other hand, the ADS we use for this case study reacts to the obstacles by stopping and waiting for the other cars to pass, so its trajectory should not change significantly. Therefore, we expect the TTO to be similar in test cases:

$$TTO(move(P, p_A, p_B, S, O_f)) \simeq TTO(move(P, p_A, p_B, S, O_s)) \qquad (\texttt{MR2}_{\texttt{TTO}})$$

MRIP$_3$: Reversed path. Following the MRIP from Equation 7.15 in Section 7.2.2, we define MRs where the origin and destination points p_A and p_B are swapped.

This change should result in the vehicle traversing the same trajectory in reverse, which would not affect the TTO significantly, thus:

$$TTD(move(P, p_A, p_B, S, O)) \simeq TTD(move(P, p_B, p_A, S, O)) \qquad (\texttt{MR3}_{\texttt{TTD}})$$

For the same reason, the TTO should also remain similar:

$$TTO(move(P, p_A, p_B, S, O)) \simeq TTO(move(P, p_B, p_A, S, O)) \qquad (\texttt{MR3}_{\texttt{TTO}})$$

MRIP$_4$: Fewer guidance points. We derive MRs form the MRIP from Equation 7.17 in Section 7.2.2, in which some of the waypoints are removed from the map ($P_f \subset P_s$). We assume that the map contains non-essential waypoints which help the vehicle navigate more accurately, similar from the blue waypoints shown in Figure 7.2. In this case, the TTD is expected to remain similar, since the trajectory should not change much:

$$TTD(move(P_f, p_A, p_B, S, O)) \simeq TTD(move(P_s, p_A, p_B, S, O)) \qquad (\texttt{MR4}_{\texttt{TTD}})$$

where $P_f \subset P_s$.

For the same reason, the TTO should also remain similar:

$$TTO(move(P_f, p_A, p_B, S, O)) \simeq TTO(move(P_s, p_A, p_B, S, O)) \qquad (\texttt{MR4}_{\texttt{TTO}})$$

Experimental Setup

Test case template. The test cases used for our experiments are short-scenario test cases with an average duration of 2 minutes (simulation time). These scenarios are based on a template city map with 51 reference waypoints provided with the open-source implementation of the SUT. The template also contains two additional cars that move in a straight line with a constant speed, which can act as obstacles for the ADS if they cross its path.

Source test cases. The source test cases were randomly generated based on the template described above. For each test case, two random points ware selected as p_A and p_B, and the two additional cars which act as obstacles were given random trajectories and speeds. The actual path that the vehicle will follow is computed by the navigation controller from the ego car by calculating the shortest path between p_A and p_B. Following this process, we generated 100 source test cases for our experiments.

Follow-up test cases. The follow-up test cases for MRIP1 (faster vehicles) were generated by multiplying the nominal speed of the vehicle by a constant. For this MRIP, we generated three groups of MRs with different speed multipliers: MR1.1 with a value of 1.1, MR1.2 with a value of 1.2, and MR1.3 with a value of 1.3. For MRIP2 (additional obstacles), the trajectory and speed of one of the other vehicles was modified to ensure that it would interfere with the ego car's trajectory. The follow-ups for MRIP3 (reversed path) were simply implemented by swapping the initial and destination waypoints. As for MRIP4 (fewer guidance points), 20% of the waypoints from the vehicle's path were randomly selected and removed, excluding the initial and destination points. In total, 600 pairs of source and follow-up test cases were generated: $3 \cdot 100$ for MRIP1, 100 for MRIP2, 100 for MRIP3, and 100 for MRIP4.

Mutants. We employed the same mutation testing approach as for the previous experiment in order to assess the effectiveness of our approach. 20 mutants of the ADS were generated by manually seeding faults in a uniform manner. We used traditional mutation operators [B$^+$12] for Simulink models in order to introduce these faults, and we checked that there were no equivalent mutants or models that could not be executed. The 100 source test cases and 600 follow-up test cases were executed in the original SUT, as well as the 20 mutants, resulting in a total of $(100+600) \times (1+20) = 14,700$ individual test executions. Some of the test executions on the original system did not terminate correctly and had to be timed out, since the vehicle failed to stop completely and close enough to the destination point. These test cases have been ignored in our evaluation,including the corresponding executions of these test cases on the mutants. In total, 4 out of 100 source test cases and one of the follow-ups for MRIP2 failed to terminate correctly, so there are $100 - 4 = 96$ test pairs for every MR and mutant from MRIP1, MRIP3 and MRIP4, and $100 - 4 - 1 = 95$ test pairs for MRIP2.

Thresholds. After some preliminary tests, we defined a threshold of 50% for $\mathrm{MR3}_{\mathrm{TTO}}$ and $\mathrm{MR4}_{\mathrm{TTO}}$, and a threshold of 15% for the rest of the MRs. Both of the 50% thresholds were found necessary for very short test cases (some of them were shorter than 10 seconds). In the case of $\mathrm{MR3}_{\mathrm{TTO}}$, the vehicle attempts to through the right lane of the road, which means that the sharpness of the turns may differ depending on the direction. As for $\mathrm{MR4}_{\mathrm{TTO}}$, removing some certain waypoints could make the vehicle take a shorter path and make significantly smoother or sharper turns.

Baseline

Since no other test oracle is available for the ADS case study, we have implemented a simple one based on thresholds for the QoS measures. This is inspired by the QoS oracles for the domain of autonomous vehicles proposed by [JST21], but we compute the ideal thresholds for our test suite instead of performing a human study. Our thresholds define maximum values for the $\frac{TTD}{distance}$ and $\frac{TTO}{distance}$ of every test case. In this context, the ideal thresholds are the smallest possible values which result in zero FPs with our test suite. Hence, these ideal thresholds are simply calculated as:

$$max_{t \in T}(\frac{TTD_t}{distance_t}) \qquad (\texttt{Threshold}_{\texttt{TTD}})$$

$$max_{t \in T}(\frac{TTO_t}{distance_t}) \qquad (\texttt{Threshold}_{\texttt{TTO}})$$

where T is our experimental test suite, which includes all the source and follow-up test cases.

Analogously to the approach followed for the definition of MRs, we also increased the thresholds by 10% to allow small variations in the performance measurements, which would be a necessary step if the thresholds were to be used realistically.

Table 7.4: Baseline results on the ADS [AVS+22a]

Thresholds	Metric	MS (%)		FDR (%)		FP (%)
Tolerance	TTD	90	95	6.89	16.12	0.00
	TTO	60		9.76		0.00
Perfect	TTD	90	95	7.02	16.74	0.00
	TTO	65		10.25		0.00

Table 7.4 shows the results obtained by these threshold-based oracles with the same test suite we use to evaluate our MRs. The “Tolerance" rows show the results obtained with the calculated thresholds increased by 10%, and the “Perfect" rows show the results for the exact thresholds. Note that the “Perfect" results are the best results that can be obtained with these oracles and test suite, since reducing the thresholds is guaranteed to result in some FPs.

As a sanity check, we validated our test oracles with a different test suite with 100 random test cases, and we confirmed that all the QoS measures where within the thresholds with 10% tolerance. On the other hand, the “Perfect" thresholds resulted in false positives, which confirms that they are too tight to be used in practice.

Experimental Results

Table 7.5 shows the experimental results obtained from the ADS experiments, including the FP, MS, and FDR metrics. In what follows, we discuss the results.

The experiments resulted in four false positives from MR3$_{\text{TTO}}$. After investing, we found that the curves have a different sharpness when traversing them from either

Table 7.5: Evaluation results on the ADS (MS: Mutation Score, FDR: Fault Detection Ratio, FP: False Positives) [AVS+22a]

MRIP	MR	MS (%)			FDR (%)	FP (%)
MRIP1	MR1.1$_{\mathrm{TTD}}$	25	55	100	0.31	0.00
	MR1.1$_{\mathrm{TTO}}$	50			2.08	0.00
	MR1.2$_{\mathrm{TTD}}$	20			0.63	0.00
	MR1.2$_{\mathrm{TTO}}$	55			2.29	0.00
	MR1.3$_{\mathrm{TTD}}$	15			0.83	0.00
	MR1.3$_{\mathrm{TTO}}$	55			2.86	0.00
MRIP2	MR2$_{\mathrm{TTD}}$	25	80		1.05	0.00
	MR2$_{\mathrm{TTO}}$	80			7.79	0.00
MRIP3	MR3$_{\mathrm{TTD}}$	90	100		14.58	0.00
	MR3$_{\mathrm{TTO}}$	65			11.98	4.17
MRIP4	MR4$_{\mathrm{TTD}}$	85	85		3.18	0.00
	MR4$_{\mathrm{TTO}}$	55			2.71	0.00

direction due to the vehicle always driving on the right lane. Usually, such cases would be compensated by having balanced right and left turns, and in very short test cases the difference would not be significant. All the test cases with FPs were found to be longer test cases (~30 seconds duration) with unbalanced left and right turns, resulting in an accumulated divergence of the TTOs. This issue could be worked around by either increasing the tolerance threshold for this MR or using only test cases where the ratio of left and right turns is not very unbalanced.

Regarding the global effectiveness, the combined MS for our MRs was 100% (20 out of 20 mutants killed). As for the FDR, the combined result was 4.19% (964 out of 23,000 metamorphic test failures). Note that we employ 20 mutants, 12 different MRs, and 100 test pairs per MR, but 4 of the source test cases did not terminate, and neither did one additional follow-up test case from MRIP2. Thus, the number of metamorphic tests is $20 \times 12 \times 96 - 20 \times 2 \times 1 = 23,000$.

Table 7.5 shows that the MS of individual MRs ranged from 15% to 90%, whereas the FDR ranged from 0.31% to 14.58%. MR3$_{\mathrm{TTD}}$ and MR3$_{\mathrm{TTO}}$ appear to be the best MRs by a wide margin, as they obtain a MS of 100% when combined together while having the highest and second highest FDRs by far. Note, however, that MR3$_{\mathrm{TTO}}$ has a non-negligible rate of FPs. On the other hand, MR2$_{\mathrm{TTO}}$ is the only other MR with comparable results, and also the MR based on TTO with the highest MS by far. Considering that there could be failure modes which can only be detected with TTO, MR2$_{\mathrm{TTO}}$ might also be a useful MR for this system.

Comparing these results with those from the oracles based on global thresholds for the QoS measures, which were presented in Section 7.3.3, our MRs killed the last mutant which could not be detected with that approach. Both approaches have identical costs in this case, so the cost-effectiveness of the MRs can be said to be better than the baseline. The mutant which could not be killed with the baseline approach and our evaluation test suite is detected by MR2$_{\text{TTO}}$, MR3$_{\text{TTD}}$ and MR4$_{\text{TTD}}$. The baseline approach does obtain better FDRs than most MRs, but the top three MRs with the best MS do obtain comparable results. Also note that one of the top MRs did yield some FPs, whereas the baseline did not.

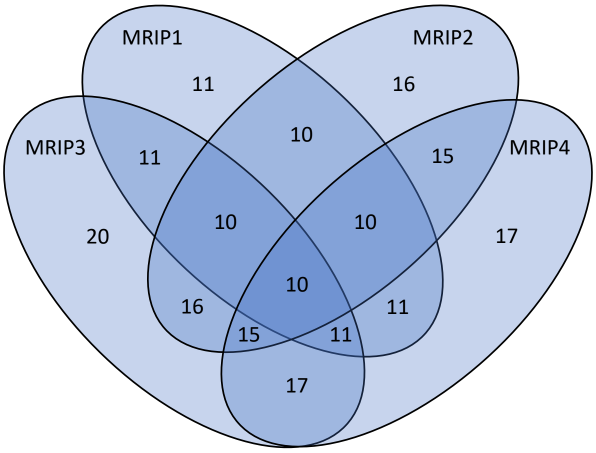


Figure 7.6: Mutants killed per MRIP (out of 20) [AVS+22a]

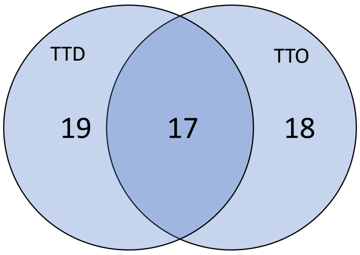


Figure 7.7: Mutants killed per QoS metric (out of 20) [AVS+22a]

Figure 7.6 shows the number of mutants killed by the MRs grouped by MRIP, whereas Figure 7.7 shows the MSs grouped by QoS metric. Regarding Figure 7.4, MRIP1, MRIP2 and MRIP4 appear to be redundant based on the results from this experiment, since MRIP3 alone killed all the mutants. However, the FP rate from MR3$_{\text{TTO}}$ might indicate that some MR from a different MRIP might be more practical to use instead of it. On the other hand, Figure 7.7 reveals that TTD is a more effective metric to detect mutants than TTO in general, especially considering that most of the killed mutants for TTO are from MR2$_{\text{TTO}}$, which is an outlier.

7.3.4 Discussion

In this section, we discuss the results obtained from both experiments and we answer our RQs based on them.

RQ1: False Positives

One of the most important factors which determine the feasibility of adopting an automated test oracle is the rate of FPs it yields. Every FP issued by an oracle is a false alarm which will cost some resources to check and dismiss, so a high rate will result in too much overhead for the oracle to be considered useful.

In order to mitigate this, we employed tolerance thresholds on some of our MRs, which can reduce the FP rate at the cost of limiting the Fault Detection Capability (FDC). Despite this, some of the MRIPs we identified from the elevator control system (Section 7.2.1) had to be changed or discarded. Compared with our initial experiments [ASA$^+$20], MRIP1 (additional calls) was changed so that the additional passenger is always the last one to arrive, because otherwise this passenger could affect the scenario in unpredictable ways, which makes identifying simple and useful MRs complicated. On the other hand, MRIP3 (faster elevators) [AVS$^+$22a] replaced MRIP4 (initial position change) [ASA$^+$20], because the latter was also found to have unpredictable effects on the QoS measures. Finally, MRIP5 (shift floors) was found to be impractical due to it changing the distance to the ground floor, which can affect the behaviour of most elevator dispatching algorithms in some cases.

Our evaluation resulted in 3 FPs in the elevator control system experiments, and 4 FPs in the ADS experiments. Considering that our MRs obtained mutation scores ranging between 88% and 100%, this rate of FPs would seem to be affordable in practice, as the benefits greatly outweigh the extra cost of checking a few more short test executions than strictly necessary.

It is also worth noting that 2 out of 3 FPs from the elevation case study did reveal some abnormal behaviour in the system, although these cases are currently dismissed by the system engineers during manual testing. These cases could potentially be handled by simply adding a duplicate issue identification system to the approach, with which these cases could be automatically marked as "ignore" or similar. In this regard, Lampel et al. proposed an approach for classifying failures based on test case features [LJAZ21] which might be applicable.

As for the remaining 5 FPs from our experiments, no abnormal behaviour was identified, and were instead caused by the inaccuracy of the MRs. These inaccuracies could possibly be worked around by employing statistical metamorphic testing [GM07], which applies MT over statistical distributions of the QoS metrics rather than using individual measurements. Nevertheless, despite being potentially less error-prone than our tolerance thresholds, this approach may not be feasible in practice due to the high cost of executing enough test cases for a meaningful statistical analysis

RQ1 – In summary: Our MRs triggered some false positives, but the number is manageable. Some of them could be avoided by detecting duplicate issues, while others may require the use of more costly statistical techniques.

RQ2: Effectiveness

Regarding the effectiveness of our approach, which we measure mainly based on the Mutation Score (MS) and Failure Detection Ratio (FDR) evaluation metrics, the experimental results show that the PV pattern is very effective at alleviating the oracle problem. Our MRs detected 88.76% and 100% of the injected faults in our case studies, which we believe justifies the cost of executing multiple test cases that is inherent to MT, especially when compared with manual testing. It is also noteworthy that most of our MRs are highly reusable across different system configurations, and even different systems with similar features. This last point is very relevant for Orona's elevator control systems, since there are different dispatcher algorithms and configurations for every building, which limits the use of techniques such as QoS value thresholds or regression oracles in new installations.

It is also worth noting that the experiments presented in this work employ random testing, which is the most basic approach for generating test cases. Using a test suite generated with more sophisticated techniques would most likely improve the results we obtained, since the failures which can be detected depend on the test suite's ability to reveal them.

Compared with the baselines, which are regression and threshold-based oracles, our MRs outperformed them in terms of MS in both experiments, showing that the PV pattern identify failure modes that are difficult to detect with regular oracles. On the other hand, the FDR of the threshold-based oracles from the second experiment was found to be higher. However, the MRs with the highest MS did outperform the FDR from the baseline. Furthermore, neither of the baselines yielded FPs, whereas our MRs did, which would result in some unnecessary extra testing effort. Nevertheless, the cost of the regression oracles from the first experiment was also much higher than that of MT. Some work has already been done in order to reduce the cost of the regression oracles in this domain by employing Machine Learning (ML) models which predict the QoS of the system without having to execute it, which removes the need to execute the reference implementation [AAI$^+$21, GAA$^+$22]. Since this ML approach is an approximation of regression testing, it still lacks the flexibility of our MRs, and its effectiveness can not be expected to be higher than the regression oracles either.

All in all, our approach seems to be more effective than the baselines as long as a small number of FPs can be tolerated. We must also remember that our MRs are applicable across different SUT configurations, whereas both regression and threshold-based would need reference executions for every configuration.

RQ2 – In summary: Our MRs based on the PV pattern are able to detect 9 out of every 10 faults in CPSs on average, which make them an effective and fully automated solution to the test oracle problem. Compared with regression and threshold-based oracles, our approach shows clear benefits in terms of cost-effectiveness and applicability.

RQ3: Differences between MRIPs and performance metrics

As for the differences between MRIPs and QoS metrics, we observed a significant disparity in terms of our evaluation metrics. When grouping the MRs by MRIPs or QoS metric, we found that some groups were subsumed by others, indicating that they might not be useful in practice.

In the experiments from the elevation domain, one of the MRIPs dominated the rest by a large margin, to the point where using any of the other MRs might not seem cost-effective. Nevertheless one of the other MRs obtained outstanding FDR results, which might make it useful to complement the best MRIP.

As for the ADS experiments, a single MRIP also dominated over the others, but one of its MRs also yielded all the 4 FPs from those experiments. Since some MRs derived from different MRIPs achieved comparable results, using those to compliment the superior MRIP might also be reasonable. Furthermore, one of the MRs from another MRIP also obtained very good results with TTO, while none of the other MRs based on that QoS metric were outstanding.

Generally, the best results seem to be obtained when combining specific MRIPs and QoS metrics, and all of the QoS metrics seemed to be able to achieve good results with some MR in both case studies, so none of them can be said to be useless. On the other hand, our results seem to imply that some MRIPs can be discarded without sacrificing much effectiveness. Ideally, some preliminary experiments would be done first in order to determine the effectiveness of each MR and select an optimal subset of them for maximum cost-efficiency. Lacking such an evaluation, we advocate for defining *diverse* MRs in terms of the type of changes that are made to the inputs and the QoS metrics that are used, since such diversity has been found to be the best way to maximize the cost-effectiveness of MT [LKTC13].

RQ3 – In summary: The results from some MRIPs and QoS metrics are subsumed by others, but there are also synergies between some of them. In line with previous results in metamorphic testing, MRs should be as diverse as possible.

7.4 Threats to Validity

In this section we enumerate the internal and external validity threats to our work, and how they have been mitigated.

7.4.1 Internal Validity

Internal validity threats refer to the factors that might have affected the results of our evaluation. One such threat is our use of mutation testing and the potential issues with it, such as the selected mutants biasing the results or the number of mutants being insufficient. For the ADS case study, we employ an amount of mutants similar to other research in the literature that uses Simulink models [LNB$^+$17, MNBB18, AWSE19]. As for the case study from the elevation domain, we employed even more mutants. We also checked for equivalent mutants in both cases and confirmed that none had identical behaviours, as recommended by Papadakis et al. [PJHLT15, PHH$^+$16]. The faults were seeded manually in both cases due to technical limitations that prevented the use of automated tools, and we introduced the mutations uniformly throughout the relevant parts of the system's source code or model in order to avoid introducing bias.

Another potential threat are the employed test suites, which might also introduce bias or contain too few test cases. Similar to the mutants, we mitigate this by generating a large and diverse set of test cases automatically. For the ADS case study, we even significantly increased the number of test cases with respect to earlier work [Val21] in order to provide a more thorough evaluation.

In the earlier stages of this work, we employed somewhat complex MRs which utilized tolerance thresholds determined by consulting with domain experts, which might introduce an additional threat [ASA$^+$20]. However, in the latest version of our work, which we present in Section 7.3, we employ much simpler MRs and thresholds in order to demonstrate that the results obtained can be replicated without resorting to a costly trial and error process.

7.4.2 External Validity

External validity threats relate to the generalizability of the obtained results. Our evaluation assesses the effectiveness of applying the PV pattern on two different case studies, which might not be enough to conclude that it is an effective approach for all or most CPSs. Nevertheless, we must note that both case studies are significantly different, both are highly complex, and the elevator control system is an industrial CPS employed in many real elevator installations.

Apart from that, the manual generation of MRs might be too complicated or costly in some cases, making this approach impractical. As mentioned before, we deliberately simplified the MRs so the results we obtain represent what can be accomplished with just a reasonable amount of domain knowledge.

7.5 Related Work

7.5.1 Metamorphic Testing

This chapter proposes the use of performance-based MRs in order to mitigate the test oracle problem in CPSs. The content of this chapter is based on our latest published research on the topic [AVS+22a]. That publication is, in turn, an extension of our previous work on applying this type of MRs to the elevator control system [ASA+20] and the ADS [Val21].

Performance MRs were already being proposed as early as [CCC+07], where Chan et al. defined a MR to detect non-functional failures on a wireless sensor network based on the power consumption of the nodes. However, this type of MR have not been widely discussed until Segura et al. explicitly defined the concept of performance metamorphic testing and pointed out its potential in many contexts [STDRC17, STDRC18]. Research on this type of MRs is being widely adopted more recently, with practical applications in domains such as data analytic platforms [JJB+19] and code generators [BBSB20]. To the best of our knowledge, our work is the first to apply performance metamorphic testing in the domain of CPSs.

Regarding ADSs, which are lately a popular SUT in verification-related research, MT has already been applied for testing various types of vehicles. Lindvall et al. proposed several MRIPs similar to the ones we use for the ADS, such as altering the path of the vehicle in a way which should not significantly affect the outcome or adding obstacles to the vehicle's path [LPMS17]. On the other hand, many publications have adopted MRIPs which simulate different environmental conditions, such as fog or rain, in order to detect erroneous behaviour [TPJR18, ZS19]. Nevertheless, all of these propose MRs based on the system's functional output or internal state, whereas our work presents the novel approach of using output relations based on performance metrics.

7.5.2 Testing CPSs

Verification and validation is one of the most important and costly aspects of CPS development, and is therefore a determining factor for the success of these products.

Because of this, there are many recent publications discussing this topic, and several of them address the test oracle problem in some way.

Regarding specified oracles, the use of different forms temporal logic has been widely researched in order to define properties of the CPSs in terms of time or space-continuous signals. Menghi et al. proposed a method to generate online test oracles for Simulink models based on properties expressed in Restricted Signals First-Order Logic (RFOL), a language which can handle continuous signals while being feasible to implement in practice [MNGB19]. Boufaied et al. developed a taxonomy of the different state-of-the-art temporal logics which can be used for the definition of test oracles for CPSs, and discussed their different levels of expressiveness [BJB+21]. Our work focuses on using MT in order to alleviate the test oracle problem in cases where defining such properties for CPSs is not feasible.

Different types of derived oracles have also been proposed for the cases where specified oracles cannot be used. A popular approach is to employ machine learning techniques in order to automatically learn invariants of the system [BHM+14]. In this regard, Chen et al. employed a Support Vector Machine (SVM) classifier trained with execution traces displaying normal and abnormal (e.g. seeded software faults) system behaviours in order to detect anomalies in a water purification plant testbed [CPS18]. Another possible application of machine learning is generating a model that can predict the system outputs, which can then be used for performing regression testing against the real system. Shahamiri et al. presented one such approach by using Artificial Neural Networks (ANNs) [SKIH11, SWKIH12], which is capable of emitting quantitative verdicts based on the error between the real and predicted outputs. Some of our work related with this thesis has also applied machine learning in the domain of elevation in order to detect [AAI+21] and non-functional [GAA+22] faults by predicting the QoS of the system. We compared the cost-effectiveness of this technique with our MT approach in Section 7.3.2. In the context of verifying Deep Neural Network (DNN) based vehicles, Zhang et al. [ZZZ+18] and Stocco et al. [SWCT20] presented different approaches for predicting unsupported driving scenarios based on Generative Adversarial Networks (GANs) and autoencoders respectively. A different approach for the autonomous vehicle domain is finding the correlation between the quality metrics used in the domain and the human perception of driving quality by means of a human study,and then generate test oracles that approximate human oracles based on quality metrics [JST21]. Other types of derived oracles, such as those based on the system's documentation, have also been proposed for software verification, but not applied in the context of CPSs yet [BHM+14]. This thesis focuses specifically on MT, which is also a technique to generate derived oracles.

7.6 Conclusion

This chapter presents a performance-driven approach for CPSs based on MT. To this end, we describe the PV pattern and derive multiple MRs from it to illustrate it. The purpose of this MRP is to facilitate the identification of MRs in different domains in order to alleviate the test oracle problem. We evaluate the effectiveness of PV by deriving and assessing MRs for two different case studies, one of which is an industrial elevator control system and the other is an open-source autonomous car. Our MRs managed to detect 88.76% and 100% of a set of faults seeded on these systems, while keeping the ratio of false positives at 4% or less. For the open-source case study, we provide a publicly available replication package of these experiments [AVS$^+$22b]. Given that the MRs are highly reusable (e.g. across different configurations of the systems), we conclude that the approach is a cost-effective solution to the test oracle problem, despite the potential difficulty of identifying the MRs and the cost of having to execute multiple test cases for checking each MR.

The lines of future work include, on the one hand, the application of this MRP on additional CPSs from different domains in order to further assert its effectiveness. On the other hand, the full potential of this approach remains to be seen, since we do not employ state-of-the-art test case generation, selection or prioritization techniques in our evaluation. Apart from that, the fault localization from a metamorphic test failure could potentially be automated, for which approaches based on metamorphic slices have already been proposed [XWCX13]. Furthermore, the identification of MRs could also be at least partially automated, for which we present an approach in Chapter 8.

Chapter 8

Automatic Generation of Metamorphic Relations

Chapter 7 has shown that Metamorphic Testing (MT) is an effective technique to mitigate the oracle problem in Cyber-Physical Systems (CPSs). However, manually defining valid and effective Metamorphic Relations (MRs) is a costly and error-prone task which requires the involvement of domain-experts who have practical experience with the System Under Test (SUT) [SFSRC16].

This chapter presents an automated approach to generate MRs based on a set of system executions labelled as correct or incorrect. We propose a genetic programming technique which co-evolves two populations of MRs with the goal of reducing their false positives and false negatives over the provided dataset of system executions. This technique is inspired by GASSERT, an approach for automatically generating Java program assertions [TJTP20, TJTP21].

Our first implementation of this approach, GASSERTMRS, is a tool to automatically generate the output relations for performance MRs. Our evaluation of this approach is based on Orona's multi-elevator installation, and compares the effectiveness of the automatically generated MRs with the ones we identified manually in Chapter 7 [ATA$^+$21, ATA$^+$22].

GENMORPH is a second implementation of the approach which can generate MRs for Java methods in a fully-automated way. In order to automate the collection of correct and incorrect executions, this tool leverages well-established solutions for test generation and mutation testing, namely EVOSUITE [FA11] and MAJOR [Jus14] respectively. The MR generation process has also been extended to generate both input and output relations automatically, i.e., full MRs. Furthermore, we also employ OASIS [JCHT18], an oracle assessment tool, in order to filter out invalid MRs. We have evaluated this solution on 10 different methods from the Apache Commons Math library in order to assess its effectiveness.

8.1 Introduction

Software systems are becoming more complex over time, making test automation an increasingly important but challenging task. This is particularly the case for Cyber-Physical Systems (CPSs), which encompass physical processes and software components and have strict safety and robustness requirements [BG11, Alu15, LS16]. One of the fundamental challenges for test automation is the so-called test oracle problem, the problem of distinguishing between correct and incorrect system behaviours [BHM$^+$14]. While there have been significant advances in automated test input generation [FA11, PKT17, LKF20], the test oracle problem is still a major bottleneck in the path towards full test automation [HMSY13].

Metamorphic Testing (MT) is an alternative technique for alleviating the oracle problem by leveraging relations among the inputs and outputs from *multiple* test executions [CCY98]. These properties, known as Metamorphic Relations (MRs), have been shown to exist in most non-trivial software systems, and are therefore a useful solution for the cases where traditional test oracles cannot determine the correctness of system executions [SFSRC16].

MT has been successfully applied in the domain of CPSs, e.g., for testing wireless sensor networks [CCC$^+$07], autonomous drones [LPMS17], or self-driving cars [TPJR18, ZS19]. This technique has also been used to define oracles for performance testing [STDRC17, STDRC18, BBSB20, JJB$^+$19]. Recently, large companies such as FACEBOOK [ABB$^+$21], GOOGLE [DL16, DELT17], ADOBE [WTZC18], and NASA [LGÁW15] are employing MT to test their systems.

While the usefulness of this technique is well recognized, identifying MRs is still a costly activity which requires extensive experience in the application domain [SFSRC16]. The automated identification of MRs is still a largely unexplored problem, and only a few solutions have been proposed thus far. MRI [ZCH$^+$14] and AutoMR [ZZC$^+$19], which are approaches to automatically identify polynomial MRs in mathematical programs, are currently the closest to a generic solution. They both rely on a search-based technique based on Particle Swarm Optimization (PSO) to effectively explore the space of possible MRs, and have been shown to effectively identify valid and useful MRs. Nevertheless, more research towards complementary and orthogonal solutions to this problem needs to be done.

We present an approach inspired by recently proposed techniques to automatically generate pre/post conditions [MPAF21] and program assertions [TJTP20, TJTP21]. Our search-based approach uses co-evolutionary multi-objective algorithms to explore the space of candidate MRs driven by a fitness function that rewards candidates with fewer false positives and false negatives. In the context of test oracles, false positives represent correct system executions in which fail but should pass, and false negatives represent incorrect executions which pass but should fail [JCHT16, JCHT18, JCHT19]. A high-quality oracle is one with no false alarms (i.e. yields no false positives) that is effective at exposing faults (yields as few false negatives as possible).

This chapter presents two versions of our automated MR generation approach, the first targeting CPSs and the second for Java methods:

- **GASSERTMRS: Genetic ASSERTion improvement for MRs** [ATA+21]. An approach to automatically generate MRs from user-provided Metamorphic Relation Input Patterns (MRIPs) and datasets of correct and incorrect system executions. We present an evaluation of this tool on and industrial case study, namely Orona's multi-elevator system. Our experimental results show that our automated approach matches or outperforms the MRs we identified manually in Chapter 7 in terms of mutation score.
- **GENMORPH: Generator of Metamorphic Relations**. An approach to generate MRs for a given Java method in a fully-automated way. We evaluate this tool on ten numerical functions from the open-source Apache Commons Math library. The results show that GENMORPH can generate effective MRs for eight of these subjects.

The rest of this chapter is structured as follows: Section 8.2 describes the GASSERTMRS approach and presents its experimental evaluation. Section 8.3 describes the GENMORPH approach and presents its experimental evaluation. Section 8.4 points out the threats to validity, Section 8.5 highlights the related work, and Section 8.6 concludes the chapter.

8.2 GAssertMRs

GASSERTMRS is our first version of the approach, which aims to automate the generation of *output relations* for MRs, and is applied in the domain of performance testing for CPSs.

8.2.1 Approach

GASSERTMRS is an adaptation of GASSERT [TJTP20], an automatic test oracle generation and improvement approach. In GASSERT, candidate solutions are Boolean expressions composed of numerical and Boolean variables, which represent program assertions. GASSERTMRS adapts this approach to generate output relations of MRs instead. The oracle improvement process requires and initial assertion (Boolean expression) and a set of correct and incorrect program states. In the case of generating oracles from scratch, a trivial expression (e.g. $true$) can be used as the initial assertion. The evolutionary algorithm explores the space of possible assertions with a co-evolutionary algorithm guided by fitness functions that reward solutions with fewer false positives and false negatives. The algorithm returns an *improved* assertion with the lowest possible number of false positives and false negatives (with respect to the correct and incorrect states from the input), in that order of priority. Assertions with zero false positives are favoured because false alarms result in costly debugging processes.

MRs are Boolean expressions that predicate on the input/output relations of test cases, instead of predicating on internal variables like assertion oracles. GASSERTMRS uses the same oracle generation and improvement algorithm as GASSERT, adapting it to the context of MT. We now describe the differences between GASSERT and GASSERTMRS.

First, GASSERTMRS does not implement the iterative improvement process used in GASSERT [TJTP20], which uses OASIS [JCHT16] to evaluate the assertion oracles, find new false positives and false negatives, and relaunch the oracle improvement with this information. This is because OASIS is not suitable for assessing MRs, as it generates individual test cases to evaluate the oracles.

Second, GASSERTMRS focuses in black-box system-level testing, as opposed to GASSERT, which predicates on internal variables at arbitrary program points. This is because generating oracles for the system's internal states is not always possible or desirable in the domain of CPSs.

Furthermore, the expressions generated by GASSERTMRS predicate on the inputs and outputs of the source and follow-up test cases, as opposed to predicating over the variables of an individual test case, since they represent output relations and not assertions over individual executions.

This approach can generate arbitrary Boolean expressions, which might be excessive in the context of MRs. Indeed, many expressions over the inputs and outputs of the source and follow-up test cases may not actually define a relation between their outputs, and thus, the result might not be a MR. For this reason, we restrict GASSERTMRS to generate Boolean expressions with the following form:

$$O_f \ [operator] \ F(O_s, I_s, I_f) \tag{MR Template}$$

where O_s and O_f are the outputs obtained from the source and follow-up test case executions; I_s and I_f are the inputs of the test cases; $F(O_s, I_s, I_f)$ is the generated

Table 8.1: Features of Elevation Test Cases [ATA+21].

Feature Name	Description
ElevatorsCount	Count of elevators available
ElevatorsDistanceFloors	Sum of the distances between the positions of each elevator in the source and follow-up test cases (MRIP3 only)
ElevatorsDifferenceTime	Aproximated time it takes to traverse ElevatorsDistanceFloors (MRIP3 only)
PassengersCount	Count of passenger calls
PassengersWaitFloors	Maximum number of floors that must be traversed to reach the calling floor of each call
PassengersWaitTime	Approximated time it takes to traverse PassengersWaitFloors
PassengersTravelFloors	Distance in floors from the calling floor to the destination of each call
PassengersTravelTime	Approximated time it takes to traverse PassengersTravelFloors

numerical expression; and $[operator]$ is a relational operator (such as $=$, $\neq$, $<$, $>$, $\leq$ or $\geq$).

GASSERTMRS assumes that all the inputs and outputs have numeric types. For cases where not all inputs are numeric, which is the case of our elevation case study, domain-specific functions to reduce the inputs into numeric features should be defined. Table 8.1 shows the features we define for the test cases in the elevation domain, where the inputs are a set of elevators and their positions and a passenger calls list, as explained in Section 4.4.1.

The generated MRs check a single (numeric) output variable, and thus, multiple MRs need to be generated with different configurations of GASSERTMRS in order to verify multiple output variables. GASSERTMRS itself only generates numerical expressions in the form $F(O_s, I_s, I_f)$, and the full output relations are constructed by using the selected output variable (O_s) and relational operator ($[operator]$) passed as input parameters for the tool. This way, each individual of the populations of the evolutionary algorithm is a numeric expression $F(O_s, I_s, I_f)$, and the full output relations O_f $[operator]$ $F(O_s, I_s, I_f)$ will only be generated to evaluate the fitness functions or to output the final MR.

Evolutionary Algorithm

At the core of both GASSERT and GASSERTMRS is a co-evolutionary algorithm that evolves a population of candidate oracles as follows:

1. **Selection** samples pairs of individuals (parents) from the population, usually preferring individuals with a better fitness.

2. **Reproduction** combines the genetic material (sub-expressions) of the parents by means of crossover operators, and returns the resulting two new individuals.

3. **Mutation** modifies the new individuals by means of mutation operators with a certain probability, and adds the resulting individual to the new population.

The co-evolutionary algorithm that evolves two populations of candidate oracles in parallel, with three competing objectives:

1. Minimizing the number of false positives,

2. Minimizing the number of false negatives,

3. Minimizing the size of the expression.

The fitness function used in the first population (ϕ_{FP}) rewards solutions with fewer false positives first, while the fitness function of the second population (ϕ_{FN}) prioritizes fewer false negatives. Both populations consider the remaining objectives only in tie cases, with the size of the expression having the least priority in both cases. The two populations exchange their best individuals every few generations (migration) in order to provide potentially useful genetic material to improve the secondary objectives.

Let $FP(\alpha)$ denote the false positive rate of α, and $FN(\alpha)$ denote the false negative rate of α, where α is a candidate oracle (assertion in GASSERT, output relation in GASSERTMRS). The following definitions are used to determine dominance among individuals for each of the two populations:

Definition 1 ***FP-fitness*** ***(*** ϕ_{FP} ***)****. Given two assertions α_1 and α_2, α_1* ***dominates*** $_{\mathbf{FP}}$ α_2 *if any of the following conditions is satisfied:*

– $FP(\alpha_1) < FP(\alpha_2)$
– $FP(\alpha_1) = FP(\alpha_2) \ \wedge \ FN(\alpha_1) < FN(\alpha_2)$
– $FP(\alpha_1) = FP(\alpha_2) \ \wedge \ FN(\alpha_1) = FN(\alpha_2) \wedge size(\alpha_1) < size(\alpha_2)$

Definition 2 ***FN-fitness*** ***(*** ϕ_{FN} ***)****. Given two assertions α_1 and α_2, α_1* ***dominates*** $_{\mathbf{FN}}$ α_2 *if any of the following conditions is satisfied:*

– $FN(\alpha_1) < FN(\alpha_2)$
– $FN(\alpha_1) = FN(\alpha_2) \ \wedge \ FP(\alpha_1) < FP(\alpha_2)$
– $FN(\alpha_1) = FN(\alpha_2) \ \wedge \ FP(\alpha_1) = FP(\alpha_2) \wedge size(\alpha_1) < size(\alpha_2)$

Next, we describe in detail the selection, reproduction and mutation operations and the specific operators we implement.

Selection. GASSERTMRS implements the following selection operators, which are chosen randomly with a configured probability every time a selection needs to be performed.

Tournament Selection. This selection operators runs two tournaments among K randomly-chosen individuals, selecting the two individuals with the best fitness as parents for the next generation [MMGG95]. Following the configuration from GASSERT, we use $K = 2$, which helps prevent getting stuck in a the local optima[Whi94].

Best-match Selection. This is a novel selection operator presented by the authors of GASSERT. This operator samples the first parent randomly from the population, and then selects the other parent according to its "complementarity" with respect to the first parent. The operator attempts to maximize their combined coverage of the correct and incorrect test executions. Here, a test execution being covered by an individual means that it is classified correctly, i.e., the expression evaluates to "true" for correct executions or "false" for incorrect executions. Specifically, a weighted random selection is used, where the weight of each individual is determined by the test executions that it covers but the first parent does not. For the ϕ_{FP}, the weights based on the coverage of correct executions, whereas for ϕ_{FN} the weights are based on the incorrect executions.

Reproduction. GASSERTMRS employs a canonical tree-based crossover as its only reproduction operator. Given two parents, a random crossover point is chosen for each parent, and then the two offsprings are created by swapping the sub-trees from the respective crossover points [Koz94].

Mutation After the reproduction step, one of the following mutation operators may be applied to the offsprings with a fixed probability. The specific operator to use is chosen randomly every time with a configured probability.

Node Mutation. This operator chooses a single node from the tree and replaces it with a new randomly generated node of the same type [BB07].

Subtree Mutation. This operator chooses a sub-tree from the tree and replaces it with a new randomly generated sub-tree of the same type [BB07].

Constant value mutation. This operator changes the value of a single constant within the tree. Specifically, a single numeric constant node is chosen, and a random value in $(-\Delta, \Delta)$ is added to its value. We define Δ as a relative small constant (we use 0.1 in our experiments) so that the constant values change in small increments. This is a new mutation operator that was not used in GASSERT, but we identified that adjusting constant values was particularly important in the context of performance testing CPSs.

Note that GASSERT used additional crossover and mutation operators that are not used in GASSERTMRS because they are inadequate for MRs.

Test Cases Classification

In order to compute the fitness of individuals GASSERTMRS employs a dataset of *correct* and *incorrect* system executions which can be used to compute *false positives* and *false negatives* respectively. Given the context of black-box testing, the elements of this dataset are tests, which include: (1) the test case inputs, (2) the execution outputs, and (3) the classification (*correct* or *incorrect*). Furthermore, in the context of MT, these tests should actually be *test pairs*.

Using *mutation testing*, the test executions from the original system can be classified as *correct*, whereas the executions from mutants with different outputs from the original can be classified as *incorrect*. Mutant executions with equivalent outputs can be discarded, as they are redundant. For our elevation case study, we define additional constraints to classify an execution from a mutant as *incorrect*. Specifically, we define a minimum threshold for the difference between the test execution outputs, in order to avoid "incorrect" outputs which are virtually impossible to distinguish from correct ones. We employ different thresholds for each output metric, which were defined with the help from domain experts: For the Average Waiting Time (AWT), at least 20% and 6 seconds increase; For Total Distance (TD), at least 10% and 5 floors increase; For Total Movements (TM), at least 10% and 4 movements increase.

In order to classify test pairs, we employ the following rules. For *correct* test executions, we simply classify the test pairs as *correct* unconditionally. For *incorrect* test executions, we only consider test pairs incorrect if: (1) the source test case is *incorrect* and we are using the $>$ or $\geq$ operators, or (2) the follow-up test case is *incorrect* and we are using the $<$ or $\leq$ operators. This is because our performance metrics are minimization objectives, so the $>$ operator would be used to identify failures from the source test case (O_s is on the left side of the operator) and vice-versa.

8.2.2 Evaluation

This section describes our empirical evaluation of GASSERTMRS, which is based on the multi-elevator system from Orona, previously described in Section 4.4.1.

Our experiments aim to answer the following four Research Questions (RQs):

- **RQ1**: *Is* GASSERTMRS *effective at generating metamorphic relations?* We assess GASSERTMRS effectiveness based on its ability to generate output relations with (ideally) no false positives and as few false negatives as possible.

- **RQ2**: *How does* GASSERTMRS *compare with unguided search for generating metamorphic relations?* We evaluate whether or not the fitness functions provide useful guidance to generate better MRs by comparing the results from GASSERTMRS with a modified version with no fitness-based guidance.

- **RQ3**: *How does* GASSERTMRS *perform when comparing with human-defined metamorphic relations?* We compare the effectiveness of the generated MRs with the results obtained by the MRs we identified manually in Chapter 7.

- **RQ4**: *How do generated metamorphic relations compare with similarly generated regular assertions?* We compare the effectiveness of the generated MRs with regular assertions generated by a modified version of GASSERTMRS, which makes use of individual test cases instead of test pairs. As a template for the regular assertions, we use: $M \leq F(I)$, where M is a performance metric, and $F(I)$ is a function over the inputs of the test case. Just like in GASSERTMRS, only $F(I)$ is generated by the evolutionary algorithm. Furthermore, we also defined a mode with "free form" assertions, where an arbitrary Boolean expression is generated by the tool, similarly to GASSERT [TJTP20]. For the Boolean expressions, in addition to the regular elements in GASSERTMRS, we enable Boolean literals and the following additional operators: $=$, $\neq$, $>$, $<$, $\geq$, $\leq$, AND, OR, IMPLIES and IFF.

Experimental Setup

Test cases and mutants. We employed mutation testing for evaluating the proposed approach. We used the same set of mutants and test cases that were used for our work in Chapter 7, where the same type of MRs were developed manually [ASA+20].

As a reminder, the dataset consists of 1,340 different test cases: 140 source test cases, 420 follow-up test cases for MRIP1, 360 follow-up test cases for MRIP2, and 420 follow-up test cases for MRIP3. Each of these test cases were executed on the original system and 89 mutants. The mutants were generated by seeding faults based on traditional arithmetic, logical and relational operator mutations [ADH+89].

The MRIPs we defined are the following [ASA+20]:

- **MRIP1:** ***Additional calls***. An additional call is inserted to the passengers list.

- **MRIP2:** ***Additional elevators***. The number of available elevators is increased, without changing the initial positions of the originally available elevators.
- **MRIP3:** ***Initial position change***. The initial positions of all the elevators are shuffled.

Initial oracles. In order to generate the initial population for GASSERTMRS, we define a simple initial expression for each configuration. For the MR oracles, the atomic numeric expression M_s was used for all cases as right hand side of Equation (MR Template), where M is the performance metric used in the configuration. For instance, a configuration with the AWT metric as M and the $\leq$ operator would start with $AWT_f \leq AWT_s$ as the initial output relation. The rationale for this is that M_s is an element which can be expected to be present on the right side of the MR. For regular assertion oracles, we just used the constant 0 as the initial right-side expression, resulting in initial assertions such as $AWT \leq 0$. Finally, for regular assertions that do not have a fixed structure, we used $true$ as the initial assertion.

Results validation. The datasets comprise source and follow-up test pairs, or individual tests for the regular assertions baseline. In order to avoid evaluating the resulting MRs with the same test cases we used for their generation, we employed 10-fold cross validation on the dataset, ensuring that the training and testing datasets contained the same proportion of *correct* and *incorrect* test executions. We used identical partitions for all the configurations, but the regular assertions baseline used different partitions due to the samples being individual test cases rather than test case pairs. In order to account for the stochastic nature of GASSERTMRS, we repeated each of the experiments 12 times with different random seeds. In total, each GASSERTMRS configuration was executed $10 \times 12 = 120$ times.

Evaluation metrics. GASSERTMRS and the baselines were evaluated based on the following metrics.

FP and **FN** represent the percentage of false positives and false negatives over the training dataset, i.e., they represent the final (and best) fitness results obtained by the evolutionary algorithm.

DF (Detected Failures) refers to the number of failing verdicts over the mutant test executions from the testing dataset.

MS (Mutation Score) refers to the percentage of mutants detected from the testing dataset (out of 89). A mutant is considered detected by an oracle if its assertion returns false in at least one of the corresponding mutant executions.

In the cases where a given assertion caused a false positive in a test, the results from the corresponding mutant executions do not affect the DF or MS. In the case of

Table 8.2: GASSERTMRS Configuration Parameters [ATA+21].

Parameter Description	Value	Parameter Description	Value
bound on the size of the assertions	32	time budget (minutes)	15
size of each of the populations (N)	1,000	prob. of crossover	90%
minimum number of generations	100	prob. of mutation	30%
maximum number of generations	10,000	prob. of tournament selection	50%
frequency of elitism (every X gen)	1	prob. of best-match selection	50%
frequency of migration (every X gen)	10	prob. constant mutation min	5%
number of assertions for elitism	10	prob. constant mutation max	50%
number of assertions to migrate (M)	160	increase prob. const. mut. every gen.	0.45%

an error occurring during the assertion evaluation (e.g. division by 0), a failing verdict is assigned.

Configuration. Table 8.2 shows the parameter values used for our experiments with GASSERTMRS. The Unguided variant used for RQ2 also used the same parameter values, except for the parent selection, which was 100% random instead, and the elitism and migration, which were disabled. These parameter values were largely based on those used by GASSERT in their experiments [TJTP20]. One difference is that the probability of either using or altering a constant during mutation was made dynamic, increasing it linearly in newer generations in order to encourage the exploration of optimal constant values in the later generations.

We configured GASSERTMRS with each of the 3 different MRIPs used in our case study, combined with 3 performance metrics, and 2 relational operators ($\geq$ and $\leq$). Hence, there were 18 different configurations of GASSERTMRS (3 MRIPs × 3 metrics × 2 operators). Furthermore, each configuration was run 12 times with different random seeds.

Results

Table 8.3 shows the evaluation results for GASSERTMRS (RQ1), as well as the ones for the Unguided (RQ2) and Manual MRs (RQ3) baselines. The reported numbers represent the median obtained on the testing dataset across each cross-validation partition and random seed. For fair comparison, the results of Manual MRs were also calculated separately over the testing sub-samples of each of the cross-validation partitions.

Table 8.4 shows the evaluation results for the Regular Assertions (RQ4) baseline, where every evaluation metric reported is again the median of over the cross-validation partitions and random seeds.

Table 8.3: Evaluation Results for RQ1, RQ2 and RQ3 (median) [ATA+21].

MRIP	Metric	Operator	GAssertMRs (RQ1)				Unguided (RQ2)				Manual MRs (RQ3)			
			FP	FN	DF	MS	FP	FN	DF	MS	FP	FN	DF	MS
MRIP1	AWT	≥	0.00%	95.99%	16.5	11.80%	0.00%	97.22%	9.0	7.87%	0.00%	92.13%	13.0	11.24%
		≤	0.00%	88.20%	28.0	14.61%	0.00%	90.00%	18.0	13.48%	0.00%	90.03%	14.0	14.61%
	TD	≥	0.00%	89.09%	12.0	12.36%	0.00%	89.77%	10.5	11.24%	0.00%	90.39%	12.5	13.48%
		≤	0.00%	92.50%	**30.5**	13.48%	0.00%	94.19%	**19.0**	8.99%	0.00%	98.93%	**4.5**	5.06%
	TM	≥	0.00%	**66.67%**	**20.0**	16.85%	0.00%	**75.00%**	**12.0**	11.24%	0.00%	**88.13%**	**7.5**	7.87%
		≤	0.00%	93.54%	9.0	4.49%	0.00%	92.00%	5.0	2.25%	0.00%	100.00%	0.0	0.00%
MRIP2	AWT	≥	0.00%	93.56%	**11.0**	**7.87%**	0.00%	96.19%	4.0	4.49%	0.00%	100.00%	**0.0**	**0.00%**
		≤	0.00%	65.20%	74.0	19.10%	0.00%	62.77%	74.0	19.10%	0.00%	64.72%	74.0	18.54%
	TD	≥	0.00%	96.70%	9.0	**8.43%**	0.00%	98.59%	4.0	3.37%	0.00%	100.00%	0.5	**0.56%**
		≤	0.00%	**82.29%**	**47.5**	12.36%	0.00%	84.52%	51.0	15.73%	0.00%	**96.57%**	**8.5**	9.55%
	TM	≥	0.00%	100.00%	2.0	1.12%	0.00%	100.00%	0.0	0.00%	0.00%	96.56%	1.5	1.12%
		≤	0.00%	**76.92%**	**22.0**	8.99%	0.00%	**83.87%**	**15.0**	5.62%	0.00%	**100.00%**	**2.0**	2.25%
MRIP3	AWT	≥	0.00%	90.57%	**28.5**	**13.48%**	0.00%	94.03%	**18.0**	8.43%	0.00%	98.51%	**1.5**	**1.12%**
		≤	0.00%	**85.85%**	**50.0**	**19.10%**	0.00%	88.62%	42.0	17.98%	0.00%	**100.00%**	**0.0**	**0.00%**
	TD	≥	0.00%	89.01%	**58.0**	**17.98%**	0.00%	94.19%	**24.0**	11.24%	0.00%	98.93%	**1.5**	**1.69%**
		≤	0.00%	**89.35%**	**30.0**	**12.36%**	0.00%	89.65%	30.0	12.92%	0.00%	**99.57%**	**0.5**	**0.56%**
	TM	≥	0.00%	**80.00%**	**16.0**	7.87%	0.00%	**90.40%**	**5.0**	2.25%	0.00%	**100.00%**	**0.5**	0.56%
		≤	0.00%	**51.92%**	**19.0**	2.25%	0.00%	54.17%	19.0	2.25%	0.00%	**100.00%**	**0.0**	0.00%

Table 8.4: Evaluation Results for RQ4 (median) [ATA+21].

Metric	Operator	GASSERT (RQ4) FP	FN	DF	MS
All	Free form	0.00%	97.40%	34.0	14.61%
AWT	≤	0.00%	91.33%	83.0	21.35%
	Free form	0.00%	96.08%	42.5	16.85%
TD	≤	0.00%	95.91%	33.5	10.11%
	Free form	0.00%	96.64%	28.0	8.99%
TM	≤	0.00%	92.45%	8.5	3.37%
	Free form	0.00%	93.94%	8.5	2.25%

RQ1: Effectiveness of GASSERTMRS. Column GASSERTMRS of Table 8.3 shows the median results obtained by our proposed approach for each of the possible configurations. While the median of FPs is 0 and the mean was always less than 1, there were actually some FPs in some configurations. It is also important to note that the results shown in Table 8.3 are obtained with the testing dataset, which represents only 10% of the test cases, and therefore the mutation score is expected to be low. For reference, the Manual MRs obtain an average mutation score of over 30% with each of the 10 cross-validation partitions of the full test suite.

The results show that GASSERTMRS is able to generate MRs with zero or very few FPs and are capable of identifying some of the seeded faults. On the other hand, we observe a very large percentage of FNs in many configurations, which may indicate that it is difficult to generate MRs which cover a large amount of seeded faults at once. Furthermore, the results show significantly different performance across the different configurations, indicating that some MRIP, performance metric, and operator combinations might be more effective than others.

RQ1 – In summary: The automatic MR generation process of GASSERTMRS can generate valid and effective MRs that have zero or very few false positives and identify some of the system failures.

RQ2: Comparison with Unguided Search. The Unguided columns from Table 8.3 show the results obtained with the evolutionary approach without fitness guidance. A comparison with the GASSERTMRS columns reveals that the latter approach dominates the former, as the results from all metrics are either the same or better. There is a single outlier configuration where Unguided obtained a higher DF, and two outlier configurations where it obtained a higher MS. The difference was minimal in most of these cases.

We also employed statistical tests to evaluate whether there is a statistically significant difference between both approaches. First, we applied the Shapiro-Wilk test to assess the distribution of the data, and concluded that it was not normally distributed. Therefore, we applied the Mann-Whitney U-test to evaluate the statistical difference. We used a significance threshold of 0.05 for the p-value, and employed the Vargha and Delaney $\hat{\mathrm{A}}_{12}$ metric in order to measure the effect sizes [VD00]. To make the results more readable, as suggested by Romano et al. [RKC+06], we categorized the $\hat{\mathrm{A}}_{12}$ values as **negligible** if $d < 0.147$, **small** if $d < 0.33$, **medium** if $d < 0.474$, or **large** if $d >= 0.474$, where $d = 2 \cdot |\hat{\mathrm{A}}_{12} - 0.5|$.

Regarding MS, the results from GASSERTMRS were better than unguided in 16 out of 18 configurations. Out of these 16 cases, 11 were statistically significant, and the difference was large in 2 cases, medium in 2 cases, and small in 9 cases, while the remaining were negligible. On the other hand, Unguided obtained a better MS in one of the two remaining configurations, with a small effect size.

For the FNs, the comparison was in favour of GASSERTMRS in 17 out of 18 cases, with 9 of them having statistical significance. 7 of those cases had a small effect size, and 3 of them had a medium effect size. As for the FPs, Unguided showed better results than GASSERTMRS in all cases, but the results were statistically significant only in 6 out of 18 cases, and the effect size was small in four cases and negligible for the remaining 14 cases.

This indicates that the guidance provided by the implemented fitness functions and selection operators does improve the generated MRs substantially, but at the cost of a small increase in the tendency to get FPs. This may be caused by the fitness guidance generating tighter MRs, which reveal more failures but may have a higher tendency for false alarms (overfitting the training dataset).

> **RQ2 – In summary:** The fitness guidance allows the evolutionary algorithm to converge into more effective MRs in many configurations, but also results in a very small increase of false positives.

RQ3: Comparison with Manual MRs. The Manual MRs columns from Table 8.3 show the results obtained with the MRs identified manually in Chapter 7. A comparison with the GASSERTMRS columns reveals that the latter approach dominates the former, as the results from the DF and MS evaluation metrics are either the same or better, barring a few outliers where there is a small difference in favour of the manual MRs. Furthermore, the differences are very significant, with a 10 times increase in MS in some cases, for instance. However, the manual MRs do have 0 false positives along the full dataset, whereas GASSERTMRS MRs do have a small but non-zero number of FPs (but a median of 0 FPs for all configurations). Note that comparing FNs is not meaningful, since the manual MRs were not generated with a process involving the training set of test executions.

Overall, we observe only two advantages for the manual MRs over GASSERTMRS: (1) exactly 0 FPs on all evaluation tests, and (2) lower complexity of the MR expressions, with a maximum of 6 elements for manual MRs and a maximum of 32 for GASSERTMRS.

RQ3 – In summary: The MRs generated with GASSERTMRS are more effective at detecting failures than those generated manually, but are also slightly more prone to false positives, as well as more complex.

RQ4: Comparison with Regular Assertions. We compare the GASSERTMRS results from Table 8.3 with the results from its Regular Assertions variant in Table 8.4. Here, we can see that there are several configurations where GASSERTMRS outperforms the regular assertions in terms of DF and MS. We disregard FNs because are not directly comparable with the MRs, since they use individual test executions instead of test pairs.

For oracles based on the TD and TM metrics, the MRs achieve better results (higher DF and MS) with the best configurations of GASSERTMRS than with the best configurations for regular assertions. Nevertheless, the regular assertions with the best results (AWT with the $\leq$ operator) do outperform any other MR configuration in both DF and MS.

On the other hand, regular assertions also achieve a median of 0 FPs, but similarly to MRs, the actual number of FPS is not always 0.

As noted before, the regular assertions employ individual test cases, and therefore each of the configurations in Table 8.4 used all the 1340 test cases in the dataset. As for the MR configurations, each of them could only use the source test cases and the follow-up test cases of their corresponding MRIP, and the follow-ups for different MRIPs were unusable. This is an inherent disadvantage of MT, and so we compare both results despite this difference. On the other hand, this allows generating more different MR oracles, because we can (and do) generate different output relations for every MRIP. this is why the MRs have more configurations. However, this also means that the effort needed for generating these oracles is higher. In this case specifically, we have 18 configurations (rows in Table 8.3) for MRs, and 7 configurations (rows in Table 8.4) for regular assertions, so the cost of generating all the MRs is $\frac{18}{7} \approx 2.57$ times higher.

RQ4 – In summary: MRs show some advantages over regular assertions, but mixed results are obtained when comparing them, with MRs being more effective with two of our performance metrics, but regular assertions being more effective with the remaining one.

Discussion

Automated generation of MR is feasible with GASSERTMRS. The need to define techniques for automatically identifying metamorphic relations has been acknowledged in a recent industrial study [ABB$^+$21]. In this paper we develop such a method based on recent advancements on automatic program assertions generation techniques GASSERT [TJTP20]. Our results show that GASSERTMRS is competitive with manually-defined MRs in the context of an industrial case study, on top of being an automated approach. According to engineers from Orona, the ability to generate MRs which are difficult to identify manually is a significant benefit of GASSERTMRS. As for the cost, manually validated field executions are often already available (e.g. for regression testing purposes), and incorrect executions can be produced by seeding faults. Thus, this approach can be adopted without unreasonable overhead.

Small test suites are sufficient for the generation of MRs in an industrial context. One of the main challenges of testing CPSs is the high cost of executing tests, which often results in having to work with a reduced test suite. Indeed, compared with the experiments with Java programs of GASSERT [TJTP20], we employed the fixed and small test suite that we had available. A small dataset might result in the evolutionary algorithm generating poor quality MR. However, our experimental results show that GASSERTMRS can generate useful MRs with this relatively small test suite, which has a realistic size to be used in practice in our case study.

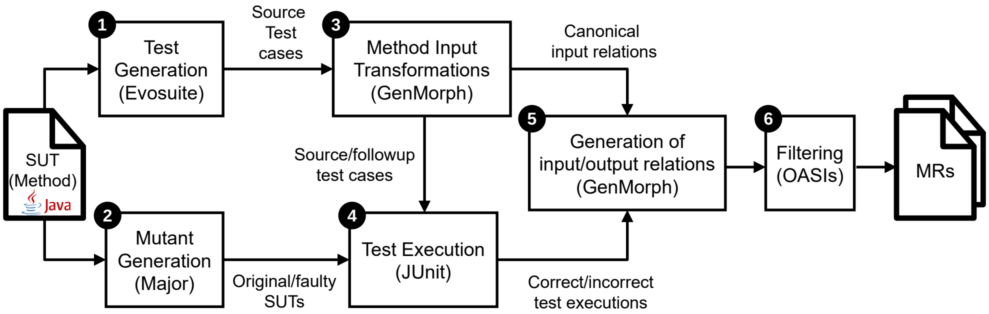
Setting constraints on the generated MRs can make the approach more effective. The current implementation of GASSERTMRS relies on numerical inputs and outputs, since the expression trees it generates only support this type of variables. We consider this constraint to be reasonable for CPSs, where the input/output domains consist of numerical variables in most cases [Alu15]. For the systems where this is not true, which is the case for our elevation case study, it is possible to transform complex inputs into numerical variables with user-defined functions that extract numeric input features. Defining these functions requires some effort, and may bias the MR generation process, but it also results in more familiar MRs and a reduced search-space. We believe that these limitations and our proposed MR template are effective for most performance testing contexts from the domain of CPSs.

Automatically generated MRs are complex. While the MRs generated by GASSERTMRS are highly effective compared with manually generated ones, many of them may be difficult to interpret for human engineers. One of the main reasons for this is the high complexity of the generated expressions, which is often partially unnecessary due to tautologies and other simplifiable sub-expressions. This issue could be mitigated by employing automated techniques that simplify expressions [Sto11] to reduce the complexity of the MRs before outputting them.

8.3 GenMorph

GENMORPH is our second version of the approach, which aims to automate the generation of full MRs, and is applied in the context of generating universal test oracles for Java methods.

8.3.1 Approach



1
Test Generation (Evosuite)
Source Test cases
3
Method Input Transformations (GenMorph)
Canonical input relations
SUT (Method)
Java
5
Generation of input/output relations (GenMorph)
6
Filtering (OASIs)
MRs
Source/followup test cases
2
Mutant Generation (Major)
Original/faulty SUTs
4
Test Execution (JUnit)
Correct/incorrect test executions

Figure 8.1: Architecture of GENMORPH MR generation

Given a function implementation P, the goal is to automatically generate one or more MRs that have zero false positives and the fewest possible false negatives. To this end, GENMORPH takes a Java method P and a time budget as its inputs. This tool explores the search-space of all the possible MRs, and yields the best ones that have been found after the time budget has expired. GENMORPH explores the search-space with an evolutionary algorithm driven by fitness functions that reward MRs with fewer false positives, false negatives, and complexity. This is a variation of the same co-evolutionary algorithm employed by GASSERTMRS and GASSERT, and similarly to those approaches, GENMORPH relies on a dataset of correct and incorrect test executions to compute the fitness of the MRs.

Figure 8.1 shows an overview of the entire MR generation process of GENMORPH, which can be broken down into six steps.

(1) Source Inputs Generation. The first step to generate a dataset for the evolutionary algorithm is to generate the test inputs. In order to discover valid and meaningful MRs, these test inputs should be diverse enough to reveal the different behaviours of the method under test. Our implementation employs EVOSUITE [FA11, FA12] to generate such a set of source test inputs. EVOSUITE is mainly driven by branch coverage, which guarantees that the test inputs cover diverse execution paths of the method under test [FA12]. Furthermore, this tool can also minimize the test suites, which compresses the dataset and reduces the execution costs of GENMORPH. The generated test inputs will be used as source test cases, as the input relations have not been decided yet at this stage.

(2) Mutants Generation. In order to obtain *incorrect* test executions, we need to execute our test inputs on faulty versions of the method under test. These faulty versions are obtained with MAJOR [Jus14], a mutation testing tool which can seed diverse artificial faults into Java methods.

(3) Initial Input Transformations. The input relations are generated based on predefined templates which specify canonical input transformations [DTPRK22]. These input transformations can be used to generate corresponding follow-up test inputs for each of the source test inputs, as well as to define the input relations. In step (5), output relations will be generated for each of these input relations, but different "relaxations" of the input relations might also be explored.

(4) Collection of Program Executions. The test executions used as a dataset for the evolutionary algorithm are generated by running the source and follow-up test input pairs. These test inputs are run on the original and faulty versions of the method under test in order to generate correct and incorrect test executions, respectively. We specifically use JUNIT to execute these tests, since it is one of the most popular testing frameworks for Java. The resulting test executions can then be used by the evolutionary algorithm to compute the false positives and false negatives of the candidate MRs

(5) Generation of MRs. GENMORPH implements an evolutionary algorithm that, given an initial input relation and a set of correct and incorrect executions, navigates the space of candidate MRs to find ones with the fewest possible false positives and negatives (in that order of priority). We propose two modes to generate MRs: *strict* and *relaxed* mode. In strict mode, we explore candidate output relations, keeping the input relation fixed, similarly to GASSERTMRS. In relaxed mode, certain "relaxations" of the initial input relation are also explored, thus allowing the generation of more general MRs.

(6) Filtering MRs. Generally, the MRs generated by step (5) will have zero false positives over the training dataset (correct test execution), since this is the highest priority of the fitness function. Nevertheless, the MRs may still be invalid, i.e., some other correct test executions may trigger false positives. To filter such MRs, GENMORPH implements a two-step filtering process that uses the RANDOOP [PLEB07] test input generator and the OASIS [JCHT18] oracle assessor.

Input Transformations

GENMORPH starts generating MRs by identifying input relations from a set of predefined input transformation templates. From the predefined transformation templates, we select a set of (metamorphic) input transformations compatible with the method under test. For each of the selected transformations, a *canonical input relation* can be derived, and the follow-up test inputs for each source test input are also generated. The canonical input relation represents the most strict interpretation of the applied transformation: Given the values of the source test inputs, there is only a single possible value of the follow-up inputs that satisfies the canonical input relation.

Our **transformation templates** are derived from the numeric MRs commonly used in the literature, which were identified by Duque et al. [DTPRK22]. Some of them can be parametrized with a number, so multiple instantiations are possible. We specifically implement the following transformations.

PermuteParameters. This transformation permutes two of the input parameters of the method. It can be applied to any pair of method parameters, as long as they are of the same type. For instance, the follow-up for $\mathit{add}(a, b)$ would be $\mathit{add}(b, a)$. In this example, the canonical input relation would be: $(a_f = b_s) \wedge (b_f = a_s)$.

BooleanFlip. This transformation flips the value of a single Boolean parameter from *true* to *false* or vice versa. Although our experimental evaluation does not contain any suitable method for this transformation, and the study of Duque et al. [DTPRK22] does not mention this relation either, we implemented it to support methods with Boolean parameters for completeness.

NumericAddition(Number). This transformation adds a constant number (positive or negative) to a single numeric parameter. For the $\mathit{add}(a, b)$ method, we could apply `NumericAddition(-1)` to its second parameter to generate the follow-up input $\mathit{add}(a, b - 1)$. The canonical input relation would be: $(a_f = a_s) \wedge (b_f = b_s - 1)$.

NumericMultiplication(Number). This transformation multiplies a single numeric parameter by a constant number (positive or negative). For the $\mathit{add}(a, b)$ method, we could apply `NumericMultiplication(2)` to its first parameter to generate the follow-up input $\mathit{add}(a \cdot 2, b)$. The canonical input relation would be: $(a_f = a_s \cdot 2) \wedge (b_f = b_s)$.

For most Java methods, there will be many possible input transformations that can be generated following the transformation templates described above. In fact, a single numeric parameter in the method results in an almost unlimited number of possible input transformations, since *NumericAddition* and *NumericMultiplication* can be instantiated with any numeric constant.

Because of this, it is necessary to sample a meaningful set of values for the parameterized transformation templates. To this end, GASSERTMRS collects a pool of constants, consisting of the predefined values -1 and 1, plus all the constant values appearing in the method under test. These constant values are extracted by instrumenting all the variable accesses and literal expressions of the method under test, and analysing their values for all the executions with the test inputs generated by EVOSUITE. If a variable access or literal expression evaluates to and identical numeric value on every execution, that value is considered meaningful for the method under test, and added to the constants pool. A weighted random sampling is then performed to select the constants to use with the parameterized templates, using the frequency with which each constant appears as its weight.

Evolutionary Algorithm

GENMORPH employs an extended version of the co-evolutionary algorithm used by GASSERT [TJTP20] and GASSERTMRS [ATA^{+}21], previously described in Section 8.2.

The core algorithm is still a co-evolutionary algorithm that evolves two populations of candidate oracles in parallel, with three competing objectives:

1. Minimizing the number of false positives,
2. Minimizing the number of false negatives,
3. Minimizing the size of the expression.

Function **MR-GENERATION** of Algorithm 1 defines the co-evolutionary algorithm used to explore the search-space of MRs. Remember that the fitness function ϕ_{FP} prioritizes false positives over false negatives, while ϕ_{FN} prioritizes false negatives. This is a version of the GASSERT [TJTP20] algorithm in which **SELECT+REPRODUCE** has been extended to support transformations of MRs at different scopes, namely, input relations, output relations, or both (whole MR).

Strict and relaxed modes. GENMORPH implements two different strategies for MR generation: *strict* and *relaxed*. In *strict* mode, the canonical input relation is not modified at all, and the evolutionary algorithm only explores the space of possible

```
input  : $R_i$: canonical input relation
         correct $\mathcal{E}^+$ and incorrect $\mathcal{E}^-$ test executions
output : $\mathbb{MR}$ a set of the best metamorphic relations
function MR-GENERATION
    $Popul^{FP}$ ← GET-INITIAL-RANDOM-POPULATION($R_i$)
    $Popul^{FN}$ ← GET-INITIAL-RANDOM-POPULATION($R_i$)
    $gen \leftarrow 0$
    repeat
        $gen \leftarrow gen + 1$
        do in parallel
            $Popul^{FP}$←SELECT+REPRODUCE($Popul^{FP}$, $\phi_{FP}$, $gen$)
            $Popul^{FN}$←SELECT+REPRODUCE($Popul^{FN}$, $\phi_{FN}$, $gen$)
        end
        if $gen$ % FREQ_MIGRATION = 0 then
            add GET-BEST-MRs($Popul^{FN}$, $\phi_{FN}$) to $Popul^{FP}$
            add GET-BEST-MRs($Popul^{FP}$, $\phi_{FP}$) to $Popul^{FN}$
        end
    until time budget is expired
    return GET-BEST-MRs ({$Popul^{FP} \cup Popul^{FN}$}, $\phi_{FP}$)
function SELECT+REPRODUCE
    $Popul$ ← COMPUTE-FITNESS($Popul$, $\phi$, $\mathcal{E}^+$, $\mathcal{E}^-$)
    $Popul_{\text{NEW}}$ ← GET-BEST-MRs ($Popul$, $\phi$)
    repeat
        $\langle MR_{p1}, MR_{p2} \rangle$ ←SELECT-PARENTS($Popul$, $\phi$)
        case mode = "strict" do
            scope ← "output relation"
        end
        case mode = "relaxed" do
            scope ← GET-RANDOM-SCOPE("output relation", "input relation", "whole MR")
        end
        $\langle MR_{o1}, MR_{o2} \rangle$←CROSSOVER+MUTATION(scope, $MR_{p1}$, $MR_{p2}$)
        add $\langle MR_{o1}, MR_{o2} \rangle$ to $Popul_{\text{NEW}}$
    until $Popul_{\text{NEW}}$ is full
    return $Popul_{\text{NEW}}$
```

Algorithm 1: GenMorph Co-Evolutionary MR Generation

output relations, similar to GAssertMRs. On the other hand, *relaxed* mode allows for certain modifications to the input relation. More specifically, this mode allows modifications to the input relation which make it strictly more satisfiable, i.e., the set of test pairs that satisfy the input relation must be a superset of the ones that satisfy the canonical input relation. The main advantage of *relaxed* mode is that it allows for the exploration of MRs with more general input relations. On the other hand, it also entails the following disadvantages wrt *strict* mode: (1) More general input relations may lead to over-generalization and false positives; (2) The search-space becomes significantly larger, and the fitness functions are also more costly to compute; (3) The *relaxed* input relations may be satisfied by multiple follow-up inputs, so generating follow-ups is not as straightforward.

Selection. GenMorph employs the same selection operators as GAssertMRs, namely *Tournament Selection* and *Best-match Selection* (see Section 8.2.1).

Reproduction (output relation). GenMorph employs the same canonical tree-based crossover [Koz94] as GASSERTMRS for the output relations (see Section 8.2.1).

Mutation (output relation). GENMORPH employs the same mutation operators for output relations as GASSERTMRS, namely *Node Mutation*, *Subtree Mutation* and *Constant Value Mutation* (see Section 8.2.1).

***Relaxed* mode operations.** In *relaxed* mode, GENMORPH can modify the input relations with specific crossover and mutation operators. There are two different ways in which the input relation can be relaxed. First, the clauses from the input relation can be dropped, such that the value of one of the input parameters for the method becomes unconstrained. For instance, the input relation $a_f = a_s \wedge b_f = b_s$ may become just $a_f = a_s$. The second way is to relax the operator used in the clauses. Currently, all the template-based input relations are a conjunctive group of equality clauses, and the equality ($=$) operator can be relaxed by replacing it with $\leq$ or $\geq$. For instance, the input relation $a_f = a_s \wedge b_f = b_s$ may become $a_f \leq a_s \wedge b_f = b_s$. In order to support these operations, we represent the input relations as the combination of: (1) A bitset indicating whether the clauses are used or dropped, and (2) a list of the operators ($=$, $\leq$ or $\geq$) used by each clause.

Reproduction (input relation). For the input relations, GENMORPH implements a single-point crossover operator [Koz94]. Crossover is performed at the same point for both the used/dropped bitset and the set of operators.

Mutation (input relation). As mentioned before, GENMORPH implements two different input relation mutation operators. On the one hand, *Clause selection* randomly selects one of the clauses from the input relations and flips its corresponding bit, indicating whether the clause is used or dropped. On the other hand, *Operator mutation* randomly selects one of the clauses and changes its operator (possibly switching $\leq$ or $\geq$ back to $=$).

Reproduction (whole MR). GENMORPH supports two crossover operators for combining the genes from two MRs. On the one hand, *Single Crossover* randomly selects either the input or the output relation, and then it randomly selects and applies an appropriate crossover operator to it. On the other hand, *Swapping Crossover* recombines the input and output relations from both parents (i.e. combines the input relation from parent$_1$ with the output relation from parent$_2$ and vice-versa).

Mutation (whole MR). GENMORPH mutates whole MRs by randomly selecting either the input or the output relation, and then randomly selecting and applying an appropriate mutation operator to it.

Constraints. GENMORPH constrains the generated MRs in order to avoid producing

```
@Test
public void testMR0() {
  int a_s = 1, b_s = 2;                    // source input
  int a_f = 1, b_f = 1;                    // follow-up input
  int o_s = add(k_s, e_s);                 // run source
  int o_f = add(k_f, e_f);                 // run follow-up
  if ((a_f == a_s) && (b_f == b_s - 1)) {  // IR is true
    assertTrue(o_f != o_s);                // check OR
  }
}
```

Listing 8.1: Executable MR Test

undesirable results. If a newly-generated individual violates either of these constraints, it is deleted immediately. First of all, the output relations have a configurable complexity limit, which the generated expressions must not surpass. In *relaxed* mode, there is an additional constraint: The input relation must be satisfied by less than 50% of all the possible test pair combinations, so as to avoid input relations that are too general, or even trivial to satisfy. Furthermore, we also implemented an “elitist” constraint which is required for an individual to be considered for the elite set or as the best individual, but is not necessary for being part of the population. This constraint is that the output relation must contain both the source and the follow-up variables from the method outputs. The goal of this is to filter out individuals which do not compare the outputs from both test cases, and are therefore not really MRs (e.g. an assertion over only the source test case). Since these individuals may still contain useful genetic material, they are still allowed into the population.

Filtering MRs

The final step of GenMorph consists of a filtering process to discard *invalid* MRs and avoid reporting them. The validity of an MR is determined by the lack of false positives after a 2-step filtering process.

The first step validates the MRs with test inputs unseen by the evolutionary algorithm. These inputs are automatically generated by Randoop [PLEB07], which is a different tool than the oue we use to generate the training dataset. In order to generate the corresponding follow-up test inputs for the automatically generated MRs, we employ a Satisfiability Modulo Theory (SMT) solver to find follow-up test inputs which satisfy the input relations. More specifically, we convert the source test inputs and the input relations to SMTLIB2 [SMT22] language constraints, and we use the Z3 theorem prover [MB08] to find follow-up input values which satisfy all the constraints.

For each MR, we generate a JUnit test suite with a test case for each source and follow-up input pair generated by Randoop and Z3, respectively. Listing

8.1 shows a JUnit test case generated for the example *add*(a, b) method and the `NumericAddition(-1)` transformation, with the output relation *add*$(a, b-1) \neq$ *add*(a, b).

The second step uses OASIS [JCHT16, JCHT18], an oracle assessor which can identify false positives and negatives for test oracles using a search-based test input generation approach. To detect false positives, the oracle assertion expression is negated and converted to a new branch, and then OASIS attempts to cover that branch in order to find counterexamples to the assertion.

OASIS is designed to work with regular program assertions, but we adapted its implementation for MRs. Given a MRs, OASIS creates a new method that, similarly to Listing 8.1, has an *if* statement with the input relation as its condition and another statement with the output relation as its body. However, unlike Listing 8.1, the generated method is not input specific, but takes the source and follow-up inputs as parameters. Furthermore, the output relation becomes a condition for an empty *if* statement rather than an assertion. This new branch becomes the target of the search-based test generation. This setup allows the detection of false positives, as OASIS has to generate inputs that satisfy the input relation but do not satisfy the output relation in order to cover the target branch.

8.3.2 Evaluation

This section describes our empirical evaluation of GENMORPH, which is based on 10 different Java methods from the Apache Commons Math library.

Our experiments aim to answer the following four Research Questions (RQs):

- **RQ1**: *Is* GENMORPH *effective at generating metamorphic relations?*
- **RQ2**: *How does* GENMORPH *compare with other baseline strategies?*
- **RQ3**: *How do* GENMORPH*'s* strict *and* relaxed *mode compare with each other?*
- **RQ4**: *Is the filtering mode of* GENMORPH *effective at detecting invalid metamorphic relations?*

Baselines

For RQ2, we implement two additional approaches for automatic MR generation: FULLMR and INPUTFIRST. These baselines still employ the same core evolutionary algorithm from GENMORPH, but handle the generation of input relations differently from the template-based input transformation approach we propose.

FULLMR uses Algorithm 1 to directly generate both input and output relations in a completely unconstrained manner. The algorithm is modified to treat input relations exactly like output relations, enabling the use of tree-based mutation and crossover operators on them. Naturally, the input relations are constrained to not use any output variables. While this baseline can explore arbitrary Boolean expressions as input relations, and is therefore more expressive than our approach, we expect that navigating this much larger search space will complicate the exploration process.

INPUTFIRST uses a hill climbing algorithm [SG06] to search for potential input relations first, and then generates the corresponding output relations using Algorithm 1 in *strict* mode. The hill climbing considers the following objectives with a descending priority: (1) Maximize the number of satisfied test inputs (encourage test input coverage), (2) minimize the number of test pairs (penalize redundant test pairs), and (3) minimize the size of the input relation. Moreover, the input relation must satisfy between 5% and 50% of all the possible test pair combinations. The lower bound of 5% ensures that the resulting input relation is not too difficult to satisfy, whereas the upper bound prevents almost-trivial input relations from being generated. The hill climbing algorithm uses the tree-based representation and mutation operators for navigating its search-space.

Experimental Setup

Experimental Subjects. We evaluate GENMORPH on the Apache Commons Math 3 Library [Apa22], one of the most popular open-source mathematical libraries for Java. Specifically, we selected 10 methods from the library which fulfil the following criteria: (1) They contain at least 10 lines of code; (2) they are static methods (to avoid methods which use or modify states); (3) all of their parameters are numeric or Boolean; (4) they are fully-deterministic pure methods (i.e. they do not use I/O or random number generators). The ten selected methods span five different classes. The first four columns of Table 8.5 show their name, signature, LOC, and number of evaluation mutants.

Step 1: Source Input Generation. For each subject, we ran EVOSUITE [FA11] (v. 1.1.0) to generate the source test inputs. We configured EVOSUITE with the branch coverage criterion, minimization enabled, and a time budget of 5 minutes. We performed 10 runs with different random seeds and aggregated all the test cases to obtain a diverse and large set of test inputs.

Step 2: Mutants Generation. For each subject, we ran MAJOR [Jus14] (v. 1.3.5) enabling all the supported mutation operators. We automatically filtered the mutations that do not directly affect the method under test.

Table 8.5: Mean results per strategy and SUT (MS = Mutation score, PZ = Ratio of MRs without FPs, SAT = Satisfiability).

method	**signature**	**LOC**	**mutants**	**strict (GENMORPH)**			**relax (GENMORPH)**			**FULLMR**			**INPUTFIRST**		
				MS	**PZ**	**SAT**	**MS**	**PZ**	**SAT**	**MS**	**PZ**	**SAT**	**MS**	**PZ**	**SAT**
nextPrime	`int(int)`	40	20	**0.77**	0.62	6.7E-4	0.52	0.60	2.8E-1	0.57	0.85	3.3E-1	0.11	0.21	4.3E-2
isPrime	`bool(int)`	40	25	0.19	0.23	3.4E-4	**0.21**	0.35	1.7E-1	0.00	0.00	3.0E-1	0.00	0.00	4.2E-2
gcd	`int(int,int)`	46	25	**0.60**	0.81	2.4E-2	0.32	0.44	5.2E-1	0.44	0.54	5.2E-1	0.06	0.04	8.0E-2
pow	`int(int,int)`	26	10	**0.58**	0.40	4.9E-3	0.51	0.52	1.1E-1	0.00	0.00	8.6E-2	0.00	0.00	3.8E-2
stirling	`long(int,int)`	55	46	**0.31**	0.69	3.3E-3	0.11	0.85	9.5E-2	0.20	0.44	9.5E-2	0.04	0.08	2.3E-2
acos	`double(double)`	59	76	0.00	0.00	8.8E-2	**0.02**	0.02	4.2E-2	0.00	0.00	5.2E-1	0.00	0.00	4.0E-1
log10	`double(double)`	12	15	**0.04**	0.02	1.7E-2	0.00	0.00	5.2E-1	0.00	0.00	4.9E-1	0.00	0.00	1.8E-1
sin	`double(double)`	53	26	**0.43**	0.29	2.2E-2	0.00	0.00	4.8E-1	0.00	0.00	4.0E-1	0.00	0.00	1.3E-1
sinh	`double(double)`	93	123	**0.13**	0.12	1.8E-2	0.02	0.04	4.9E-1	0.01	0.02	4.2E-1	0.00	0.00	-
tan	`double(double)`	58	37	**0.32**	0.33	3.0E-2	0.00	0.00	5.3E-1	0.00	0.00	4.2E-1	0.00	0.02	2.5E-2

Step 3: Initial Input Transformation. For both the *strict* and *relaxed* modes of GenMorph, we select four instantiations of the input transformation templates for each run.

Step 4: Collection of Program Executions. We enforced a limit of 9,000 correct and 9,000 incorrect executions in order to prevent slowing the evolutionary algorithm down by providing too large of a dataset. The test executions are randomly sampled if there are more than those configured amounts.

Table 8.6: Configuration Parameters for Algorithm 1

Parameter Description	Value	Parameter Description	Value
time budget (minutes)	30	prob. of crossover	90%
max correct executions	9,000	prob. of mutation	30%
max incorrect executions	9,000	frequency of migration (every X gen)	10
bound on the size of R_o	16	number of individuals for elitism	10
size of each of the populations	1,000	number of individuals to migrate	160

Step 5: Generation of MRs. Table 8.6 shows the configuration of Algorithm 1 we use for all of our experiments. For each subject method and strategy, we used a time budget of 30 minutes and we generated four different MRs. For the inputFirst baseline, the budget was split into 10 minutes for hill climbing and 20 minutes for Algorithm 1. We repeated each run 12 times in order to account for the stochastic nature of our algorithms.

Step 6: Filtering MRs. For each subject, we ran Randoop [PLEB07] (v. 4.3.0) with 12 different random seeds. Because of the high cost of running Z3 [MB08] (v. 4.8.14), we sampled a maximum of 100 Randoop test cases for each run. Next, we executed multiple runs of OASIs with a budget of 150 seconds per run and an overall budget of 10 minutes. For each MR, the filtering process is interrupted if a false positive is found at any point

We ran OASIs [JCHT18] multiple times (due to its stochastic nature) with an overall budget of 10 minutes, giving each run a budget of 150 seconds. We stop OASIs when we find the first FP.

Results

RQ1: Effectiveness of GenMorph. RQ1 evaluates the effectiveness of GenMorph at generating effective MRs using **Mutation Score (MS)**, i.e., the ratio of mutants killed.

We only consider the MS of the MRs that pass the 2-step filtering process described in Section 31). For each MR which passed the filtering, we ran the PIT [CLH$^+$16]

(v. 1.7.4) mutation testing tool to compute the MS. We use 12 different test suites generated by RANDOOP and Z3, resulting in a total of 12 different MS measures for each MR. For MRs that did not pass the filtering step, we record a MS of 0 for each of the 12 runs. Note that our evaluation setup deliberately uses different tools for generating the MRs and for evaluating them.

Table 8.5 shows the average MS obtained with each approach and subject method. The average MS ranges from 0% to 77% for *strict* mode, and between 0% and 52% for *relaxed* mode. Excluding `acos` and `log10`, the average MS is no smaller than 13% for *strict* mode in the remaining eight methods, which can be considered effective. On the other hand, the *relaxed* mode obtains an average MS of at least 11% for five out of ten methods.

RQ1 – In summary: GENMORPH generates effective (>10% MS on average) MRs for 8 out of 10 methods, and very effective (>50% MS on average) ones for 3 out of 10 methods with its *strict* mode.

RQ2: Comparison with Baselines. RQ2 compares our MR generation approach with two baselines. We computed the evaluation metric (MS) of the MRs generated by these baseline techniques following the same process we used for RQ1.

Since all the approaches share the same source test executions, we pair their runs with the same random seeds together for comparison. Thus, we employ the paired Wilcoxon signed-rank test [Woo07] to assess the statistical significance of the difference between the MS of each pair of approaches. We set the significance threshold for the p-value to 0.05. We measure the effect sizes by employing the Vargha and Delaney $\hat{A}_{12}$ metric. As suggested by Romano et al. [RKC$^+$06], we categorized the difference as **N** (*negligible*) if $d < 0.147$, **S** (*small*) if $d < 0.33$, **M** (*medium*) if $d < 0.474$ and **L** (*large*) if $d >= 0.474$, where $d = 2|\hat{A}_{12} - 0.5|$.

Table 8.7 shows the effect sizes for the comparison between the approaches. The comparisons that are not statistically significant are omitted (-). The subscript for each effect size is 1 if the technique on the left has the better MS, or 2 otherwise.

The results reveal that *strict* mode outperforms FULLMR with a Large difference in six methods while FULLMR never outperforms *strict* mode. On the other hand, *relaxed* mode outperforms FULLMR with a Large difference in two methods, but it is conversely outperformed in three methods.

As for INPUTFIRST, it is outperformed by *strict* mode in six methods, while it never outperforms *strict* mode. *Relaxed* mode outperforms INPUTFIRST with a Large difference in four methods, and is also never significantly outperformed by INPUTFIRST.

Table 8.7: Comparison of techniques (Effect size for MS).

	S<>F	**S<>I**	**R<>F**	**R<>I**	**S<>R**
nextPrime	L_1	L_1	S_2	L_1	L_1
isPrime	L_1	L_1	L_1	L_1	-
gcd	S_1	L_1	L_2	L_1	L_1
pow	L_1	L_1	L_1	L_1	S_1
stirling	L_1	L_1	M_2	M_1	L_1
acos	-	-	N_1	N_1	N_2
log10	N_1	N_1	-	-	N_1
sin	L_1	L_1	-	-	L_1
sinh	M_1	M_1	-	S_1	M_1
tan	L_1	L_1	-	-	L_1

Legend: S = strict, R = relaxed, F = fullMR, and I = inputFirst

RQ2 – In summary: The *strict* mode of GenMorph clearly outperforms both baselines with statistical significance. The *relaxed* mode clearly outperforms inputFirst, but not the fullMR baseline.

RQ3: Comparison between Modes. This RQ aims to compare the relative effectiveness of the two GenMorph modes: *strict* and *relaxed*. We measure their relative effectiveness in terms of MS, using the same statistical analysis used in the baselines comparison from RQ2.

The **S<>R** column from Table 8.7 shows the comparison between the MS obtained with the *strict* and *relaxed* modes. The results show that *strict* mode outperforms *relaxed* mode with statistical significance with a Large difference in five methods, while the *relaxed* mode never outperforms *strict* with a non-negligible difference.

Even though Table 8.5 shows that relaxed mode obtains a better average MS for two of the methods, the difference is not significant. In general, *strict* mode clearly outperforms *relaxed*, particularly for the five methods with floating-point arithmetic (parameters of type double). For these methods, *relaxed* mode never obtained a better MS than 2%, while *strict* obtains an MS higher than 10% for three of them. The cause of this may be that it is easy to over-generalize and generate invalid MRs when inequalities are used on the input relations for these methods, which is what *relaxed* mode does.

RQ3 – In summary: The results suggest that *strict* mode, which directly uses the canonical input relations from the input transformation templates, generally achieves better results than *relaxed* mode.

RQ4: Filtering. This RQ evaluates the effectiveness of the filtering process (step 6) in detecting invalid MRs. We refer to the ratio of MRs that pass both steps of the filtering process as **PZ**.

Table 8.5 shows that the PZ ranges between 0% and 69% for *strict* mode and between 0% and 85% for *relaxed* mode. The results also indicate that the filtering process eliminates significantly more MRs for the methods that use floating-point arithmetic. A possible reason for this is that floating-point arithmetic can have special values such as NaN or Infinity, which the MR may not handle correctly if these values were not part of the dataset for their generation. For instance, the property $sinh(x+1) > sinh(x)$ may appear to be correct, but does not hold for $x = \infty$.

To provide more insights on the satisfiability of the input relations, we count the number of unique test inputs generated by OASIS, as well as the number of those unique inputs that satisfy the input relation. We call the ratio of the latter count over the former one the Satisfiability (**SAT**) of the input relation.

The SAT columns from Table 8.5 show their average values. Note that SAT values are never zero. The missing value for the last column of the *sinh* row is because all the MRs generated by INPUTFIRST were filtered during the first step, leaving none to compute the satisfiability. The results show that OASIS is more effective at generating test inputs that satisfy input relations generated with the *relaxed* mode than by the *strict* mode. These results are expected, since the goal of *relaxed* mode is to obtain more general input relations, which makes them easier to satisfy. The lower SAT for *strict* mode is not necessarily a problem, since MRs with a more general input relation have a higher chance for false positives.

RQ4 – In summary: GENMORPH's filtering process is effective at filtering MRs that do not generalize well on unseen inputs, and the MRs generated with *strict* mode are harder to falsify than those generated with *relaxed* mode.

Discussion

Automated generation of MR is feasible with GENMORPH. Following the research work from GASSERTMRS, we have developed a fully-automated approach to generate MRs, and we have shown that this approach can be effectively used to identify useful MRs for Java methods. The MR generation process has a cost comparable to GASSERT [TJTP20, TJTP21], which generates program assertions using a similar evolutionary process using samples of correct and incorrect executions.

Multiple iterations of GENMORPH might be needed. Our results have shown that GENMORPH still fails to generate a good proportion of useful MRs for some method, possibly due to factors like special floating-point values (NaN, infinity) not being considered in the training dataset. This issue might be solved by including an outer loop of the filtering process, similarly to how GAssert [TJTP20] employs an outer loop with OASIs [JCHT16, JCHT18] in order to allow further refinement of the test oracles after new false positives and false negatives have been identified. With this change, the final MRs would be expected to be significantly more robust, possibly allowing GenMorph to obtain positive results with the Java methods for which it currently generates very few useful MRs. Nevertheless, this change would also result in a significant increase of the total time budget needed for generating MRs.

The relaxing of MRs is currently ineffective. Our comparisons between the *strict* and *relaxed* modes of GenMorph have revealed that the generalization of the input relations performed in *relaxed* mode are counter-productive with the current approach. We speculate that more general MRs are more prone to false positives, resulting in fewer valid MRs passing the filtering process, and thus, worse average results. However, we must note that integrating the filtering step as an outer loop for GenMorph, as discussed in the previous point, may resolve this issue. Another concern is that the search-space becomes much larger when the input relations are not constant, and mitigating this may require the definition of more advanced exploration techniques.

Adapting GenMorph for CPSs is challenging. GenMorph is a general MR generation approach which could theoretically be adapted to the context of CPSs, but testing this type of systems brings in an additional set of challenges. One of them is the inability of our current input transformation templates to express many useful MRIPs. Indeed, looking at the MRIPs we identified for our elevation case study in Chapter 7, “Faster elevators” is the only one that could be expressed with our templates. In order to resolve this, more advanced input transformation templates which operate on sets, lists, and other complex types should be defined. Beyond functional concerns, generating a large dataset of correct and incorrect test executions might not be possible for many CPSs due to the high cost of executing tests on these systems. Some techniques, such as surrogate models, could be used to mitigate this [Jin11], but the usefulness of MRs generated based on datasets created by such means would need to be investigated. Indeed, the use of such models for testing activities when the cost of executing the real system is too high has already been proposed by previous work [MS11, BSH17, KM18]. In fact, potentially suitable Machine Learning (ML) models for predicting performance measures have already been developed for our industrial elevation CPS case study, and are currently used the context of generating regression testing oracles [AAI+21, GAA+21a, GAA+22].

8.4 Threats to Validity

In this section we enumerate the internal and external validity threats to our work, and how they have been mitigated.

8.4.1 Internal Validity

Internal validity threats refer to the factors that might have affected the results of our evaluation. One such threat may arise from errors in the measurements or the implementation of our tools. We mitigate this threat by manually inspecting the output and internal behaviour of the tools for a few sample runs.

Another potential threat to the validity of our empirical evaluation is the use of mutation testing, which might have introduced a bias in our results. Mutation testing introduces a bias when the mutants are too few or there are many equivalent mutants [PJHLT15]. To mitigate this threat, we used an amount of mutants comparable to other related works that use simulation-based mutation testing [AWSE19, LNB^{+}17, MNBB18] for the case study from the elevation domain used to evaluate GASSERTMRS. Furthermore, we checked these mutants to identify and filter out equivalent mutants, as recommended by Papadakis et al. [PJHLT15]. As for GENMORPH, we employed MAJOR [Jus14] and PIT, which are well-established mutation testing tools that can automatically generate a large amount of faulty versions of the SUT.

The test suites which might also introduce bias or contain too few test cases. For GASSERTMRS, we generated a large and diverse set of test cases automatically. As for GENMORPH, we employed existing test suite generation tools to mitigate this threat, namely EVOSUITE [FA11] and RANDOOP [PLEB07].

The approach and evaluation methods we use are also inherently stochastic. We account for this by repeating the experiments multiple times with different random seeds and performing statistical analyses. For GASSERTMRS, each configuration was executed 12 times, and 10-fold cross-validation was used to partition the test suite into non-overlapping training and testing subsets, resulting in $10 \times 12 = 120$ samples. As for GENMORPH, we ran each configuration 12 times, and we performed the evaluation with 12 different test suites, resulting in $12 \times 12 = 144$ samples.

8.4.2 External Validity

External validity threats relate to the generalizability of the obtained results. Our evaluation to assess the effectiveness of GASSERTMRS employs a single case study which might not be enough to conclude that it is an effective approach for most CPSs. Nevertheless, we note that this is a highly complex industrial CPS employed in many real elevator installations. As for GENMORPH, we mitigated this threat by selecting ten different functions from a popular open-source library. Moreover, we evaluated GENMORPH with unseen test inputs and seeded faults generated with different tools from the ones used to generate the MRs.

8.5 Related Work

Metamorphic testing for CPSs. MT has already been used to mitigate the oracle problem CPSs. Lindvall et al. combined MT and model based approaches in order to perform simulation-based testing of autonomous drones [LPMS17]. Several other works have explored the use of MT to verify self-driving cars [TPJR18, ZS19]. Unlike those works, ours focuses on the generation of MRs, rather than on their application.

NLP-based MR generation. Regarding documentation-based techniques, Blasi et al. proposed MEMO [BGE+21], an approach to automatically derive equivalence MRs from JavaDoc comments. Their solution breaks the problem into two major steps: *Finding* sentences which describe MRs in the JavaDoc and *translating* the identified MRs from natural language to an executable specification. Naturally, the quality of derived MRs depends on the completeness and correctness of the available documentation. In contrast, our approach relies on labelled test executions rather than documentation. Our solution is also not limited to equivalence MRs.

ML-based MR generation. ML has also been leveraged to identify specific types of MRs. Kanewala et al. proposed techniques to predict MRs based on the the control flow graph of the SUT using graph kernels [KB13, KBBH16]. On the other hand, Zhang et al. presented RBF-MLMR, another MR prediction approach based on an improved radial basis function (RBF) neural network [ZZPL17]. These approaches only predict whether predefined types of MRs hold for a given method, whereas our solution generates executable MRs.

Search-based MR generation. As for search-based techniques, MRI [ZCH+14] and AUTOMR [ZZC+19], which are both approaches based on PSO which can discover polynomial numeric MRs that are valid, i.e., have no false positives over a given set of correct executions. AUTOMR extended the MRI approach by adding support for: (1) Inequality MRs, (2) multiple inputs, and (3) arbitrary degree polynomials. Unlike our approach, these techniques only consider false positives as a fitness function, and therefore have no guidance to generate more effective (e.g. higher mutation score) MRs, just valid ones. Furthermore, our approach is not limited to polynomial expressions for the generated MRs.

Assertion generation. GASSERT [TJTP20, TJTP21] and EVOSPEX [MPAF21] generate oracles with evolutionary algorithms driven by false positives and false negatives, based on a dataset of test executions labelled as correct or incorrect. However, these approaches generate regular program assertions, whereas our approach aims to generate MRs.

8.6 Conclusion

This chapter presented a novel approach to automatically generate MRs, which is one of the first solutions that have been proposed for this topic [ZCH+14, ZZC+19]. Our solution employs a co-evolutionary algorithm which evolves two populations of candidate MRs, with the goal of reducing their false positives and false negatives over a given set of test executions labelled as correct or incorrect. We present two implementations of the approach: GASSERTMRS, which generates performance MRs for a given MRIP, and GENMORPH, a more complete and generic approach which fully automates the MR generation process for Java methods.

Our experiments with GASSERTMRS show that the it is possible to generate valid and effective MRs in the context of an industrial CPS, enabling the automation of a significant part of the testing process at an affordable cost. The experimental results showed that the automatically generated MRs outperformed the manually identified ones, given a time budget of 15 minutes for each run of the evolutionary algorithm.

The evaluation of GENMORPH, which was done in the context of testing Java methods, also showed that this approach can be extended to fully automate the process, with only the SUT given as an inputs, and still obtain positive results. The experiments revealed that GENMORPH could efficiently generate useful MRs for eight out of ten subject methods.

Future works will aim to increase the applicability and generalizability of the approach.

Extending GASSERTMRS. An interface to easily plug test generation and mutation testing tools would greatly enhance the experience of adopting this tool for a new system. Full automation like GENMORPH will never be feasible due to the different tools that are used in different domains. Then, GASSERTMRS should be evaluated with other CPSs in order to assess its usefulness in different application domains.

Extending GENMORPH. While the general design of GENMORPH allows for arbitrarily complex expressions and type systems, our current implementation for Java methods only supports Boolean and numeric values. More work is needed to support complex types (e.g., lists or sets) and their operations in the generated expressions, and research on how this extension affects the cost-effectiveness of the tool (search-space may become larger) needs to be done.

Adapting GENMORPH for CPSs. Except for the automation of generating correct and incorrect test executions, the GENMORPH approach is general enough to be adapted to the context of CPSs. However, there are many challenges that need to be overcome for that. One of them is the inability of our current input transformation templates to express many useful MRIPs, as they are currently only based on numeric and Boolean operations. Another challenge is the high cost of generating a large dataset of correct and incorrect test executions for CPSs, and techniques to mitigate this issue need to be explored.

Improving MRs readability. We found that the MRs generated by our approach could be simplified in many cases. Examples include tautologies (e.g. $true \vee A \Rightarrow A$) or expressions with shorter mathematical equivalents (e.g. $1 + 1 \Rightarrow 2$). An important future work is to investigate post-processing to simplify the MRs, as well as a study on how easy the generated MRs really are for the human engineers to understand.

Part IV

Optimization

Chapter 9

Metamorphic Test Selection

Part III presented Metamorphic Testing (MT) as a solution to the test oracle problem [BHM^{+}14] for Cyber-Physical Systems (CPSs), which is one of the most complicated problems in the verification process of these systems. Nevertheless, the verification process still requires other components, such as a test suite. Although it is usually simple to generate some (e.g. random) test suite for most Systems Under Test (SUTs), the cost-effectiveness of the verification process greatly depends on its quality, since it must reveal the different aspects of the system's behaviour in order to allow the test oracles to verify them. There are many existing solutions for this problem, such as techniques to generate test cases with more diverse inputs [CLM04]. However, since metamorphic tests employ *multiple* test executions, most of these solutions do not map directly into MT.

This chapter presents a generic and novel approach for metamorphic test selection which aims to optimize the cost-effectiveness of MT [AAPA22a]. We specifically explore the general case where a provided test suite contains metamorphic test cases for multiple Metamorphic Relations (MRs), and we aim to find the most cost-effective subsets. We formulate this as an optimization problem, and we propose a search-based technique to obtain a set of pareto-optimal solutions. Our approach is evaluated on the test suite from the elevation domain employed in the evaluation of Chapter 7.

9.1 Introduction

Metamorphic Testing (MT) [CCY98] is a testing technique based on relations between the inputs and outputs of *multiple* test executions, the so-called Metamorphic Relations (MRs). In this thesis, we have applied this technique in order to alleviate the oracle problem for CPSs [ASA^{+}20, AVS^{+}22a], and we have achieved great success in our experiments with both open-source ad industrial case studies. Other research has also reported positive results with this technique in other types of CPSs, such as autonomous drones [LPMS17] and self-driving cars [TPJR18, ZS19].

Testing CPSs is a very costly process [BG11], since executing the SUT is usually computationally expensive at the Software-in-the-Loop (SiL) level (using simulators), and requires setting up the physical environment and real-time executions at the Hardware-in-the-Loop (HiL) and operation levels. Furthermore, the fact that MRs require at least two test executions to verify often translate to even higher costs, since more test executions will be needed. To mitigate this issue, test selection aims to select a subset of the test suite in order to optimize the cost-effectiveness of the process.

In this chapter, we present an approach for metamorphic test selection, and we report our experiences after applying it in our industrial case study from the elevation domain. Our approach employs the NSGA-II evolutionary algorithm in order to obtain pareto-optimal solutions for the metamorphic test selection problem. We describe the problem representation we use for this, the objective functions we use to guide the search algorithm, and a set of new genetic operators for this problem. For our evaluation, we employ the execution cost and the mutation score as metrics in order to measure the the quality of the solutions. We compare our approach with Random Search and NSGA-II without our new genetic operators, and we conclude that our proposed solution is more effective than these baselines. Furthermore, we try different objective combinations with our approach, and we discuss which objectives provide the best guidance for obtaining the most cost-effective test suites.

We also provide a replication package for the experiments we present in this chapter, which includes the source code for our implementation of metamorphic test selection, the test inputs for the test suite, the mutants killed with each metamorphic test, and the results obtained after applying our test selection approach [AAPA22b].

The rest of this chapter is structured as follows: Section 9.2 presents the proposed approach, Section 9.3 defines the experimental evaluation and discusses the obtained results, Section 9.4 summarizes the lessons learned from our evaluation, Section 9.5 points out the threats to validity, Section 9.6 describes the related work, and Section 9.7 concludes the chapter.

9.2 Approach

In this section, we describe our metamorphic test selection approach.

9.2.1 Problem Formulation

We formulate metamorphic test selection as a multi-objective optimization problem in which the cost of the test suite must be minimized while maximizing its effectiveness. Specifically, we use execution time as the cost metric and Mutation Score (MS) as the effectiveness metric, although other metrics can be used with this approach. The test suite TS consists of a set of test case pairs, each of which belongs to a specific MR ($TC^m_{MR_n} \in TS$). Since the goal of this technique is to minimize the cost of executing the test suite, we assume that the test cases have not been executed yet. Thus, each test case TC consists of the system inputs only, and the cost and effectiveness of the solutions might have to be predicted or approximated based on that information alone. The goal is to compute subsets of the test suite $TS' \subset TS$ with the best cost-efficiency.

9.2.2 Search Algorithm

Considering our problem formulation, we chose a multi-objective search algorithm which can find the pareto-optimal solutions. More specifically, following other research on test selection [YH07, PWAY16, LFN+17, AWM+19], we chose the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [DPAM02] as the metaheuristic for our approach. This algorithm is known to be less effective when using more than three objectives [PKT17], but we consider this limitation to be acceptable for this problem.

9.2.3 Genetic Operators

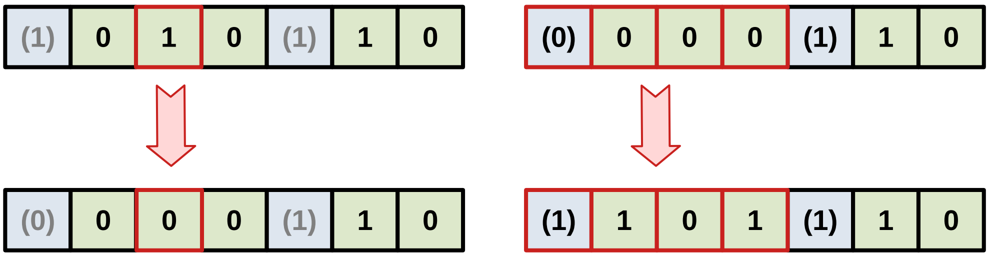
Next, we describe the solution representation we employ for this problem, and then we describe the mutation and crossover operators used on this representation. While some of these are equivalent to the traditional genetic operators for bit sets [YH07], we also define new ones which operate on groups of metamorphic tests which share the same source test case.

Solution Representation

We represent a metamorphic test selection solution as a bit set indicating whether each of the follow-up test cases has been selected or not. This representation is derived from the observation that each follow-up test case usually has a single corresponding source test case, which they are often derived from. Because of this, each metamorphic test corresponds with an individual follow-up test case and vice-versa, and thus selecting follow-up test cases and selecting metamorphic tests is equivalent. With this representation, the source test cases are implicitly selected based on whether at least one of its follow-ups is selected or not, since executing only the source or the follow-up test case is never desirable in MT. To the best of our knowledge, there is no metamorphic testing work in the existing literature which contradicts the assumption we make here (follow-up test cases gave a single corresponding source test case).

Despite the solutions only consisting of bit sets representing the selected follow-up IDs, the algorithm must globally store the test inputs for each test ID, as well as the mappings of source test cases to their corresponding follow-up test cases. This information is used by the objective functions, as well as some of the genetic operators.

Mutation



(a) Follow-up flip

(b) Source group flip

Figure 9.1: Mutation operators [AAPA22a]

- *Follow-up flip*.This operator selects or deselects a single follow-up test case from the test suite by randomly flipping a single bit from the solution. This is equivalent to a standard bit-flip mutation. Figure 9.1a shows a follow-up test case being chosen as a mutation point and getting deselected (a value of 1 indicates selected, and 0 deselected). The source test cases, which we depict as bits between parentheses with a blue background, are not actually part of the solution, but rather implicitly selected or deselected based on the corresponding follow-ups. In this example, the first source test case becomes implicitly deselected because none of its follow-ups is selected after the mutation.

- *Source group flip*. This operator selects or deselects all the follow-up test cases corresponding with a single source test case. This is done by randomly selecting a source test case from the metadata first. If the source test case is (implicitly) deselected, one of the follow-ups is randomly chosen and selected first, ensuring that the source test case is (implicitly) selected after the mutation, and then every other follow-up has a fixed probability of being selected. On the other hand, if the source test case is (implicitly) selected, all the follow-ups are simply deselected. Figure 9.1b shows the first source test case being chosen as a mutation point. One of the follow-ups is randomly chosen and always selected, and the rest of its follow-ups can be selected with a fixed probability. In this case, its first and third follow-ups are selected, but the second one is not.

Crossover

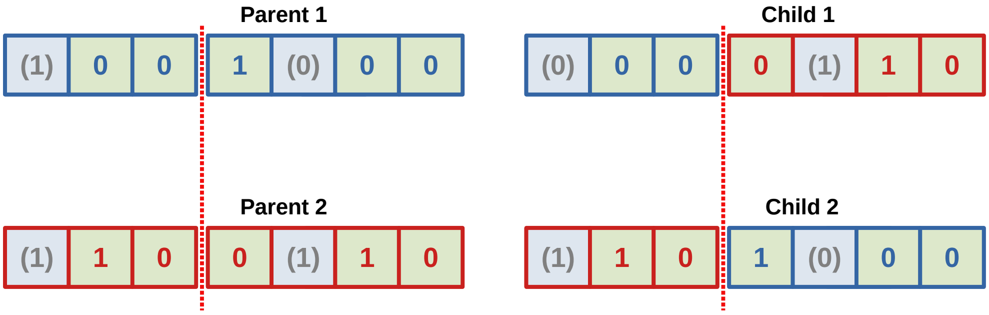


Figure 9.2: Follow-up crossover [AAPA22a]

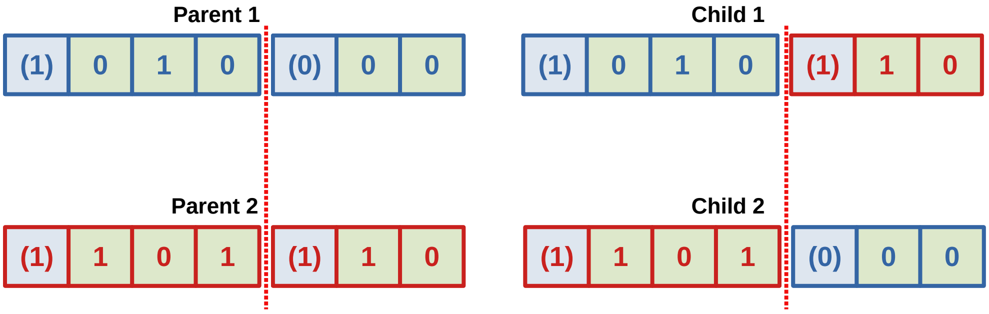


Figure 9.3: Source group crossover [AAPA22a]

- *Follow-up crossover*. This operator performs a single-point crossover on a randomly chosen crossover point over the bit set representing the solution, where each bit corresponds with a follow-up test case. This is equivalent to a standard single-point crossover. Figure 9.2 shows an example where the crossover point (dotted vertical red line) is set after the second follow-up test case, and so the follow-up selections are crossed over that point. Remember that the source test cases (bits between parentheses with blue background) are not part of the solution, and this particular operator does not interact with them directly. However, Child 1 ends up having the first source test case deselected despite both parents having it selected, because none of its follow-ups is selected after the crossover from this example.

- *Source group crossover*. This operator performs a single point crossover over the source test-case groups. Although the source test cases are not part of the solution representation, a crossover point is chosen for the source test cases, and a crossover of entire groups is performed. At the solution level, all the bits that correspond to the same source group will always be entirely copied from either parent, never combined. Figure 9.3 shows an example where the crossover point (dotted vertical red line) is after the first source test case. Child 1 ends up with Parent 1's bits for source test case 1 and Parent 2's bits for source test case 2, while Child 2 is constructed the other way around.

Multiple Operators

Since our proposed approach contains two mutation and two crossover operators, a strategy for applying them must be defined. In our implementation, we use top level mutation and crossover operators which choose one of the corresponding operators randomly with equal probability, and then apply the chosen operator. This way, for example, ~50% of the crossovers will use "Follow-up crossover", and the remaining ~50% will use "Source group crossover".

9.2.4 Objective Functions

We define a set of objective functions to guide the search algorithm towards more cost-effective solutions. Keep in mind that at the point the algorithm is run, the system outputs are not available, and thus the objective functions are based on the test inputs and their features. Even though the inputs and their features are domain-specific, many of our objective functions are generic. Other than the *Cost*, which is a minimization objective, the rest of them are maximization objectives which should approximate the test suite's effectiveness.

- *Cost.* This objective aims to minimize the execution cost of the test suites. Its value is calculated as the sum of the costs of the selected test cases. Since the real execution time is unknown, we employ the following value for the elevation domain in order to approximate the execution time in seconds for a given test case:

  $$T_{last} - T_{first} + 60$$

  where T_{last} is the arrival time of the last passenger, and T_{first} is the arrival time of the first passenger, all measured in seconds. This formula estimates that all the passengers will be transported to their destination 60 seconds after the last call to the elevators, which was found to be a good approximation after some tests. The cost estimation we use here is specific to the elevation domain, but it is generally simple enough to approximate a formula for most domains. Ultimately, the cost of each test case could just be estimated to be 1 if there is no better approximation or if all test cases have similar costs.

- *MR Coverage.* This objective aims to balance the number of metamorphic tests selected for each MR by maximizing the minimum count of tests selected for any MR, as shown in the following formula:

  $$\min_{\forall MR_n \in MR} \left(\frac{|TC_{MR_n}^{selected}|}{|TC_{MR_n}^{all}|} \right)$$

  where MR is the set of all MRs used in our test suite, $|TC_{MR_n}^{selected}|$ is the count of test cases for MR_n selected, and $|TC_{MR_n}^{all}|$ is the total count of test cases for MR_n. This is a novel objective function which has been defined specifically for metamorphic test selection, and is domain-agnostic. The rationale behind this objective is that diversifying the MRs might result in a higher failure detection capability, since different MRs may be able to detect different types of failures.

- *Input Diversity.* This objective aims to maximize the diversity of the inputs [CLM04]. The input diversity of a solution TS' is calculated as the sum of the minimum distance of each input test case from every other input test case in the test suite:

  $$\sum_{t_1 \in TS'} \left(\min_{t_2 \in TS' \setminus t_1} (dist(t_1, t_2)) \right)$$

  The $dist(t_1, t_2)$ function computes the euclidean distance between t_1 and t_2, which are numeric vectors representing the test inputs of the individual test cases (not test pairs) in TS'. Since many types of systems have complex inputs that are not wholly numeric, a vector of numeric *features* can be used instead. For instance, for the elevators system described in Section 4.4.1, the passenger calls and elevator positions are complex inputs that have different lengths on each test case, so we derive numeric features such as elevators count, passenger count, or the ratio of passengers going up/down from lower/middle/upper floors. While the input vector shapes are domain-specific, this function is otherwise domain-agnostic.

- *Passenger Density.* This objective aims to maximize the overall passenger density of the selected test cases, which is computed as the sum of the passenger counts on each test case divided by the sum of the costs of each test case:

  $$\frac{\sum_{t \in TS'} |t_{calls}|}{\sum_{t \in TS'} cost(t)}$$

  where $|t_{calls}|$ is the number of passenger calls and $cost(t)$ is the estimated cost of test case t, which uses the same formula as the *Cost* objective function. The reasoning behind this objective is that more dense test cases will result in more complex decisions taken by the elevator system, which might reveal more failures. This objective is specific to the elevation domain.

- *Passenger Count.* This objective aims to maximize the total passenger count of the selected test cases:

  $$\sum_{t \in TS'} |t_{calls}|$$

  where $|t_{calls}|$ is the number of passenger calls in test case t. The reasoning behind this objective is that more passenger calls will result in more operations from the elevator system, which would be expected to reveal more failures. This objective is specific to the elevation domain.

- *Passenger Distance Travelled.* This fitness function aims to maximize the total distance travelled by the passengers in the selected test cases:

  $$\sum_{t \in TS'} \sum_{c \in t_{calls}} |c_s - c_d|$$

  where $|c_s - c_d|$ is the distance travelled for the passenger call c. The reasoning behind this objective is that longer travel distances will result in more complex decisions taken by the elevator system, which might reveal more failures. This objective is specific to the elevation domain.

Table 9.1: Objective function combinations (*Cost* is always included) [AAPA22a]

Combination	Objective 1	Objective 2
c1	MR Coverage	-
c2	Input Diversity	-
c3	Passenger Density	-
c4	Passenger Count	-
c5	Passenger Distance Travelled	-
c6	MR Coverage	Input Diversity
c7	MR Coverage	Passenger Density
c8	MR Coverage	Passenger Count
c9	MR Coverage	Passenger Distance Travelled
c10	Input Diversity	Passenger Density
c11	Input Diversity	Passenger Count
c12	Input Diversity	Passenger Distance Travelled
c13	Passenger Density	Passenger Count
c14	Passenger Density	Passenger Distance Travelled
c15	Passenger Count	Passenger Distance Travelled

We decided to try every combination of two or three objective functions, where the first objective is always *Cost* and the remaining one or two objectives can be either of the other ones, which are used to estimate the efficiency of the test suite. This is in line with our chosen algorithm's limitations, as NSGA-II is known to not scale well beyond three objective functions [PKT17]. Hence, we have 15 possible objective function combinations, which are shown in Table 9.1.

9.3 Evaluation

This section enumerates the Research Questions (RQs) we aim to answer with our evaluation, describes our experimental setup, and presents the results obtained.

9.3.1 Research Questions

We define the following RQs for our experiments:

- **RQ1**. *How does our approach compare with Random Search (RS)?* This RQ aims to simply determine the effectiveness of our approach by comparing it with a trivial baseline. We compare the effectiveness of our approach (NSGA-II-MET) with that of Random Search (RS), for each of the 15 possible objective combinations.

- **RQ2**. *Do the new genetic operators we propose increase the effectiveness of our approach?* This RQ aims to determine whether or not our new source group based mutation and crossover operators increase the effectiveness of our approach. We determine this by comparing a variant of our approach which only uses the traditional operators (NSGA-II-TR) with our full approach (NSGA-II-MET), for each of the 15 objective combinations.

- **RQ3**. *Which of the different objective combinations is most effective with our approach?* This RQ aims to determine the best objectives to use with our approach in order to guide the algorithm towards more cost-effective solutions. In order to answer this RQ, we compare the results of all 15 objective combinations for NSGA-II-MET.

9.3.2 Experimental Setup

Algorithm Configuration

All the experiments were configured with populations of size 100 and with the number of fitness evaluations set to 25,000 in order to make comparisons of the results fair. The NSGA-II experiments were configured with binary tournament selection [DPAM02], a crossover rate of 1.0, and a mutation probability of $\frac{1}{N}$, where N is the count of metamorphic tests on the full test suite. All the other parameters were chosen based on previous work related to multi-objective test selection [YH07, WAG13, PWAY16, AWSE16a, AWM+19].

Evaluation Dataset

For this evaluation, we employed the same set of test cases and mutants as the ones presented in [ASA+20]. The metamorphic tests are based on the Metamorphic Relation Input Patterns (MRIPs) described that work, namely: MRIP1 (Additional call), MRIP2 (Additional elevators), and MRIP3 (Initial position change). The test suite consists of 140 randomly generated source test cases and 1200 follow-up test cases: 420 for MRIP1, 360 for MRIP2, and 420 for MRIP3. These are short-scenario test cases with an average duration of ~3 minutes. Mutation scores were computed over a set of 89 mutants of Orona's most common dispatching algorithm, which were generated by seeding faults based on traditional operator mutations [ADH+89]. The full test suite achieves a total mutation score of 83.1% [ASA+20].

The evaluation dataset consists of two separate groups of artifacts: (1) The set of test case inputs and the list of test input pairs for each MR, and (2) a table indicating which mutants are killed by each metamorphic test. Only the data from (1) was used for the test selection, whereas the data from (2) is used exclusively for the evaluation, since only (1) would be available before executing the entire test suite.

Evaluation Metrics

We evaluated the effectiveness of solutions using the following metrics:

- The **cost** of a solution indicates how expensive it is to execute the test suite. In this case, we measure the sum of the execution times for the selected test cases in seconds.

- The **Mutation Score (MS)** of a solution is the ratio of mutants that are identified as incorrect by at least one of the selected metamorphic tests. Test cases which have a false positive (i.e., identify the non-mutant system output as incorrect) are not considered for this metric.

The solutions provided by the search algorithm are only pareto-optimal for its objective functions, which are different from these evaluation metrics. Specifically, the exact MS is never known before executing the test suite, so the objective functions (other than cost) are used to estimate it. Therefore, a second pareto-frontier based on *cost* and *MS* is generated for the evaluation, and the solutions are translated to it. We use the *Hypervolume (HV)* of this evaluation pareto-frontier in order to measure the quality of the solutions, similar to other multi-objective test case selection work [PODPDL14, AWM^{+}19, OP21, AVAS22]. A higher HV, indicates a more effective test selection approach.

Although the HV is a widely accepted metric to evaluate multi-objective search algorithms [WAY^{+}16], it is interesting to see the effectiveness of each algorithm for different cost thresholds. This is because some approaches might, for instance, be more effective at finding solutions specifically under 10% of the total cost. Because of this, we also employ time budget based *Decision Makers (DMs)* in our evaluation to provide additional insights. These DMs receive an evaluation pareto-front as an input, and output the best solution for the given cost threshold (i.e., the solution with the highest MS at or under the specified time budget). In previous studies on this topic, industrial participants have shown interest in computing the most effective test suites for a fixed time budget [PWAY16], which is what DMs provide. In this work, we use time budgets based on fractions of the cost of the full test suite, and we refer to the DM for $X\%$ of the cost as DM_X. For instance, DM_10 provides the best MS that can be obtained with a time budget of 10% of the execution time for the whole test suite.

Statistical Tests

Due to the stochastic nature of our test selection algorithms, we repeated every experiment configuration 50 times with different random seeds, as suggested by Arcuri and Briand [AB11]. We then performed statistical tests over the results in order to analyse them. First, we applied the *Shapiro-Wilk* test [SW65] to determine the distribution of the data. Then, after confirming that the data was normally distributed, we applied the t-test to assess whether there was statistically significant difference between the algorithms.

The p-value threshold for statistical significance was set to 0.05. Furthermore, for the cases with statistical difference, we employed the Vargha and Delaney $\hat{A}_{12}$ test [VD00] to assess the effect size of the difference between approaches. To make the results more readable, as suggested by Romano et al. [RKC^{+}06], we categorized the $\hat{A}_{12}$ values as **N** (*negligible*) if $d < 0.147$, **S** (*Small*) if $d < 0.33$, **M** (*Medium*) if $d < 0.474$, or **L** (*Large*) if $d >= 0.474$, where $d = 2 \cdot |\hat{A}_{12} - 0.5|$.

9.3.3 Experimental Results

In this section, we discuss the results obtained from our experiments and we answer our RQs based on them.

RQ1: Comparison vs Random Search (RS)

This RQ aims to determine the effectiveness of our approach (NSGA-II-MET) by comparing it with a trivial baseline, namely, Random Search (RS).

Table 9.2: (RQ1 + RQ2) Statistically significant effect sizes and average improvement of NSGA-II-MET with respect to RS and NSGA-II-TR for the HV metric [AAPA22a]

Combination	NSGA-II-MET vs RS		NSGA-II-MET vs NSGA-II-TR	
	Effect size	Improvement	Effect size	Improvement
c1	L+	52.53%	-	0.32%
c2	L+	53.05%	-	-0.39%
c3	L+	41.81%	L+	14.45%
c4	L+	69.70%	L+	2.91%
c5	L+	71.62%	M+	1.36%
c6	L+	50.99%	-	0.10%
c7	L+	47.99%	L+	7.24%
c8	L+	64.87%	S+	1.22%
c9	L+	63.74%	M+	2.07%
c10	L+	51.57%	-	0.79%
c11	L+	66.42%	L+	2.37%
c12	L+	66.57%	L+	1.97%
c13	L+	60.44%	L+	4.66%
c14	L+	60.65%	L+	5.09%
c15	L+	70.05%	L+	2.37%
Average		59.47%		3.10%

Table 9.2 shows the comparison of the HV from our approach (NSGA-II-MET) and Random Search (RS). The “Effect size” column under “NSGA-II-MET vs RS” shows the $\hat{A}_{12}$ effect sizes from the statistical comparison, with a ‘+’ postfix indicating that NSGA-II-MET obtained better results, or a ‘-’ postfix indicating the opposite. The results show that our approach achieves a better HV than RS with statistical significance and a large effect size for all objective combinations.

The “Improvement” column shows the average improvement of our approach compared to RS for the HV. The final row shows an overall average improvement of ~60%, with the improvements for different objective function combinations ranging between ~40% and ~70%.

Figure 9.4 shows the average mutation scores for various DMs with the objective combination that achieved the best HV (c15). This figure reveals that RS fails to find any suitable solution for time budgets under 45% of the full cost. The HV is also lower for higher cost solutions up to 60% of the cost, although the difference is not

Table 9.3: (RQ1 + RQ2) - Statistically significant effect sizes for mutation scores with different DMs [AAPA22a]

	DM	1	5	10	15	20	25	30	35	40	45	50	55	60
NSGA-II-MET vs RS	c1	M+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L-	L-	L-
	c2	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	-	-	L+
	c3	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	M+	L+	L+
	c4	S+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+
	c5	S+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+
	c6	S+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L-	L-	M-
	c7	S+	L+	L+	L+	L+	L+	L+	L+	L+	L+	-	-	S+
	c8	-	L+	L+	L+	L+	L+	L+	L+	L+	L+	-	-	-
	c9	N+	L+	L+	L+	L+	L+	L+	L+	L+	L+	-	-	-
	c10	S+	L+	L+	L+	L+	L+	L+	L+	L+	L+	M-	M-	S-
	c11	N+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+
	c12	N+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+
	c13	S+	L+	L+	L+	L+	L+	L+	L+	L+	L+	S+	-	N+
	c14	N+	L+	L+	L+	L+	L+	L+	L+	L+	L+	S+	-	-
	c15	S+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	L+	-
NSGA-II-MET vs NSGA-II-TR	c1	-	-	L+	M+	M+	S+	-	-	-	M-	L-	L-	L-
	c2	M+	L+	M+	S+	-	-	-	S-	S-	-	-	-	-
	c3	S+	-	-	M+	L+	L+	L+	L+	L+	L+	L+	L+	L+
	c4	-	-	S+	L+	L+	L+	L+	M+	L+	L+	L+	L+	M+
	c5	-	S+	S+	-	M+	L+	L+	L+	-	S+	-	-	-
	c6	-	-	M+	S+	-	-	-	-	-	-	S-	M-	M-
	c7	-	-	M+	S+	-	M+	L+	L+	L+	L+	L+	L+	M+
	c8	-	-	M+	L+	-	-	-	-	-	-	-	-	S+
	c9	-	-	L+	L+	L+	-	S+	-	-	M+	-	-	-
	c10	-	-	-	-	-	-	-	-	M+	-	-	-	-
	c11	-	-	L+	M+	M+	S+	-	-	-	-	-	-	
	c12	-	-	L+	M+	L+	-	-	-	-	-	-	-	M+
	c13	-	-	M+	L+	M+	M+	M+	L+	M+	L+	L+	M+	L+
	c14	-	S+	M+	L+	L+	L+	L+	-	L+	M+	M+	L+	L+
	c15	-	S+	-	M+	L+	L+	L+	L+	M+	L+	L+	L+	M+

as extreme. Table 9.3 shows all the mutation score comparison across all objective combinations and DMs. This table shows that RS does not always obtain worse results at DM_50 or higher, although our approach still obtains better results for most objective combinations. Even though RS appears to obtain significantly better results at DM_50 or higher with some objectives (c1, c6 and c10), RQ3 will reveal that these objectives are the least effective ones overall, which makes that RS obtaining better results with those objective combinations is not significant.

RQ1 – In summary: Our approach was significantly more effective than RS for every possible objective function combination, with a ~60% higher HV on average.

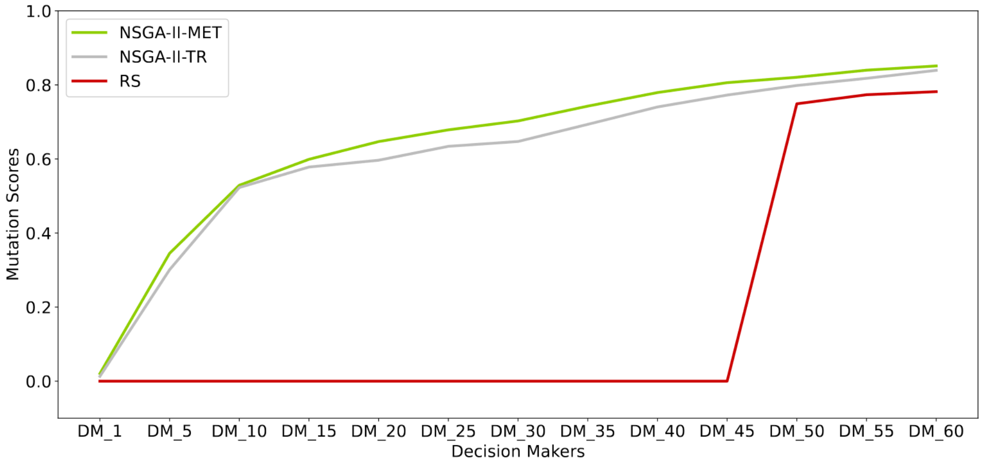


Figure 9.4: Average mutation scores for the best objective combination (c15) [AAPA22a]

RQ2: Effectiveness of Genetic Operators

This RQ aims to determine the effectiveness of our new genetic operators (NSGA-II-MET) by comparing it with a variant of our approach where only the traditional genetic operators are used (NSGA-II-TR).

The “Effect size” column under “NSGA-II-MET vs NSGA-II-TR” from Table 9.2 reveals that our approach with the new genetic operators dominates over the version without them in terms of HV. The results show statistically significant improvements with 11 out of 15 objective combinations, and no statistically significant difference with the rest of them.

As for the “Improvement” column, the final row shows an overall average improvement of ~3%, with the improvements for different objective function combinations ranging between ~0% and ~14%.

Figure 9.4 reveals a modest but consistent improvement of the mutation score for all the DMs with the best objective combination (c15). The bottom half of Table 9.3 also reveals various degrees of improvement across the board, except in some cases with the objective combinations c1, c6 and c10. Once again, these outliers are not significant because, as RQ3 will reveal, these three objectives are the least effective ones.

RQ2 – In summary: The use of our new genetic operators resulted in similar or moderately better results across all the objective combinations, with a ~3% higher HV on average, revealing that the new operators do improve the search process.

RQ3: Best Objective Combinations

This RQ aims to determine the most effective objective function combinations for our approach (NSGA-II-MET) by comparing them with each other.

Table 9.4: (RQ3) Statistically significant HV effect sizes of objective combination comparisons for NSGA-II-MET. "+" suffix indicates better HV for the objective in the first column [AAPA22a]

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15
c1	-	S+	L-	L-	L-	-	L-	L-	L-	-	L-	L-	L-	L-	L-
c2	S–	-	L-	L-	L-	S-	L-	L-	L-	M-	L-	L-	L-	L-	L-
c3	L+	L+	-	L-	L-	L+	S-	L-	L-	M+	L-	L-	L-	L-	L-
c4	L+	L+	L+	-	-	L+	L+	L+	L+	L+	S+	M+	L+	L+	-
c5	L+	L+	L+	-	-	L+	L+	L+	L+	L+	S+	S+	L+	M+	-
c6	-	S+	L-	L-	L-	-	L-	L-	L-	-	L-	L-	L-	L-	L-
c7	L+	L+	S+	L-	L-	L+	-	L-	L-	L+	L-	L-	L-	L-	L-
c8	L+	L+	L+	L-	L-	L+	L+	-	-	L+	L-	L-	-	M-	L-
c9	L+	L+	L+	L-	L-	L+	L+	N+	-	L+	L-	L-	S-	S-	L-
c10	-	M+	M-	L-	L-	-	L-	L-	L-	-	L-	L-	L-	L-	L-
c11	L+	L+	L+	S-	S-	L+	L+	L+	L+	L+	-	-	M+	S+	M-
c12	L+	L+	L+	M-	S-	L+	L+	L+	L+	L+	-	-	M+	-	M-
c13	L+	L+	L+	L-	L-	L+	L+	-	-	L+	M-	M-	-	-	L-
c14	L+	L+	L+	L-	M-	L+	L+	M+	S+	L+	S-	-	-	-	L-
c15	L+	L+	L+	-	-	L+	L+	L+	L+	L+	M+	M+	L+	L+	-

Table 9.4 shows the effect size of the HV comparisons between all the objective function combinations with each other. For the most effective objective combinations, c4, c5 and c15 stand out because they dominate the rest, as they show statistically significant improvement over some others while not being worse than any other with statistical significance. These objective combinations correspond with "Passenger Count" (c4), "Passenger Distance Travelled" (c5) and the combination of both (c15). The combination of both objectives (c15) appears to be the most effective by a small margin.

On the other side of the spectrum, the worst results are obtained by "Input Diversity" (c2), followed by "MR Coverage" (c1), and then the combination of both (c6).

Figure 9.5 shows a graph of the average mutation scores from every objective function combination for different DMs. This reveals, once again, that c4, c5 and c15 achieve better results than the other objective combinations.

In conclusion, the domain-specific "Passenger Count" and "Passenger Distance Travelled" objective functions provide the best guidance towards test suites with

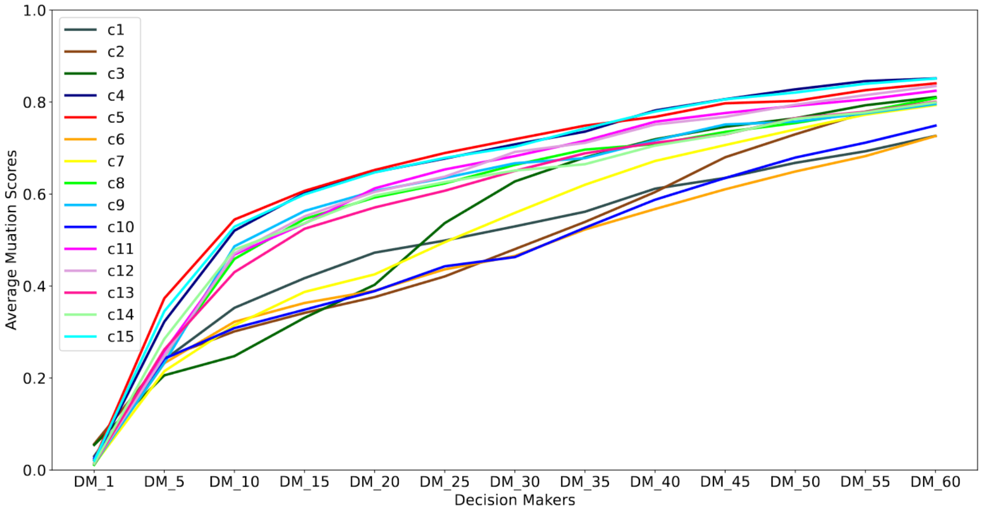


Figure 9.5: Average mutation scores for all objective combinations [AAPA22a]

higher mutation scores, whereas generic objectives such as “MR Coverage” or “Input Diversity” are clearly less effective.

RQ3 – In summary: The domain-specific maximization objectives “Passenger Count”, “Passenger Distance Travelled”, and the combination of both achieved the best results, whereas “Input Diversity” and “MR Coverage” had the worst results.

9.4 Lessons Learned

This section presents the lessons learned from this work, and highlights related future research avenues of interest.

Lesson 1: Effectiveness of the approach. Our experimental results have revealed a huge potential for reducing the cost of MT without significantly compromising its effectiveness. In our case study, looking at the average mutation scores achieved with our approach (Figure 9.4), we can achieve 90% of the mutation score from the whole test suite for only 50% of its total cost, and 60% of the mutation score can be achieved for only 10% of the total cost. These results are just the first approximation to the problem, as other fitness functions and test suites generated with different techniques still need to be evaluated, and are likely to yield even better results.

Lesson 2: Domain-specific objective functions. The results have also shown that the most effective approach is to employ some domain-specific objective functions we defined. Specifically for our case study from the elevation domain, the experiments show that maximizing the number of passengers and the distance they traverse are the best objective functions to obtain test suites with higher mutation scores. Generalizing the multi-elevator controller as a system that performs tasks, the best objective functions appear to be maximizing count and the total complexity of the tasks to perform, which makes sense. Conversely, diversifying the inputs or the MR coverage does not appear to be very effective in comparison.

Lesson 3: Different target time budgets. The results from Table 9.3 and Figure 9.5 suggest that different objective functions may be more effective depending on the target time budget. For instance, Figure 9.5 shows that c5 obtains the best mutation scores up to DM_35, but beyond that point c4 obtains better results. In this particular case, the difference between the top three objective combinations (c4, c5 and c15) is mostly negligible, so just using the overall best one (c15) for all cases would probably be acceptable. Nevertheless, using different objective functions depending on the target time budget might be a lot more worthwhile for other case studies where the top objective functions vary significantly depending on the DM.

Future prospect 1: Generalizability to other systems. Our experimental evaluation is based on a single case study, so our conclusions do not necessarily generalize to other types of systems. It would be interesting to perform a similar evaluation on CPSs from different domains to see if our approach is still effective. The selection of objective functions is a specific aspect where a large set of experiments from different domains would be required in order to provide general guidelines.

Future prospect 2: MR Coverage based on historical results. The results from our experiments have shown that our “MR Coverage” objective is not very effective for guiding the search process. Nevertheless, our implementation attempts to provide a uniformly distributed coverage for all the MRs, which may not be the best approach. In Chapter 7 of this thesis, we found that there is often a significant disparity between the effectiveness of the different MRs [ASA+20, AVS+22a], so biasing “MR Coverage” such that the most effective MRs get selected more might be a better approach. While this is not possible to do for a new set of MRs, their effectiveness can be derived from historical data as different iterations of the SUT are tested (e.g. number of bugs found by each MR). This historical data could be leveraged to enhance the metamorphic test selection process by adjusting the “MR Coverage” and refining the selected test cases accordingly. An analysis on how this tuning might be performed and how effective it can be would be an interesting research topic.

9.5 Threats to Validity

9.5.1 Internal Validity.

The test suite and mutants we employ for our experiments are a potential threat to the validity of our work. We argue that the number of mutants we use is significantly higher than other CPS testing experiments [LNB$^+$17, MNBB18, AWSE19], and that the test suite is large enough, considering the high cost of executing it (approximately 2 days and a half) [AVS$^+$22a]. The parameters chosen for the search algorithm are another potential threat. To mitigate this, we chose parameter values based on previous studies on multi-objective test case selection [YH07, AWSE19]. The parameters were also different NSGA-II and RS because many of them are not applicable to RS. Nevertheless, the maximum number of fitness evaluations were the same, making the comparisons between them fair, and the baseline algorithm (RS) has no other parameters to consider. Furthermore, we had to consider the stochastic nature of the search algorithms we employ. We deal with this non-determinism by repeating each experiment 50 times and applying statistical tests to analyse the results.

9.5.2 External Validity.

The main external validity threat relates to the generalizability of our results, since our experiments are based on a single case study. In this regard, we note that we use a complex industrial system which is used in production in many real multi-elevator installations. Still, we acknowledge that our conclusions might not generalize to all CPSs, and we propose further experiments to remedy that.

9.6 Related Work

9.6.1 Test Case Selection

Test case selection is widely recognized as a multi-objective problem [CGT$^+$20]. Since Yoo and Harman [YH07] first proposed the use of pareto-based search algorithms for test case selection, many research works have focused on applying these algorithms in different domains, such as software product lines [WAG13], Java programs [MHD15], or autonomous vehicles [BGK$^+$22]. Our work applies these technique in the context of an elevator control system from Orona.

Regarding CPSs specifically, test case selection has also been applied in various domains, such as configurable CPSs [YH07] or Simulink models [AWM$^+$19]. Our approach tackles a variation of this problem, namely, metamorphic test selection, where metamorphic test cases comprise *multiple* test cases.

Some approaches have also proposed specific operators for the search algorithm in order to enhance the process. Arrieta et al. proposed and evaluated different strategies for seeding the initial population of the algorithm [AVAS22]. Panichella et al. proposed various operators to increase the diversity of the selected test cases [PODPDL14]. Olsthoorn and Panichella proposed a novel crossover operator based on linkage learning [OP21]. We also propose new mutation and crossover operators, but ours are designed specifically for the metamorphic test selection problem.

9.6.2 Metamorphic Testing Cost Minimization

In order to reduce the cost of MT, some publications have presented different criteria for selecting the most effective MRs, and thus ignore the ones that are likely to yield worse results. Many of them simply identify features of the MRs which might indicate that they are stronger or weaker (e.g. output relations that are simple equalities are generally weaker) [CHTZ04, MG06, CZC13]. More recently, Srinivasan et al. proposed the use of coverage and fault based criteria to prioritize MRs [SK22].

On the other hand, other research has applied regular test case selection strategies for this problem. Barus et al. noted that most MT research simply employs random testing for generating the source test cases, and reported that using Adaptive Random Testing (ART) instead could improve the effectiveness of MT [BCK$^+$16]. Regarding white-box techniques, Alatawi et al. used dynamic symbolic execution (DSE) to generate source test cases with better source-code coverage [AMS16], and Saha et al. explored using various coverage-based criteria from EvoSuite [SK18]. Analogously, other work has focused on selecting the best follow-up test cases instead. In this regard, Arrieta proposed a multi-objective strategy to generate follow-ups for deep learning systems, which aims to minimize the number of generated follow-ups while maximizing the uncertainty of the model [Arr22a].

This work presents an approach for metamorphic test selection, which is more general and fine-grained than either MR selection, source test case selection, or follow-up test case selection. The former requires MRs to be either fully selected or discarded, whereas the others consider either the source or the follow-up test cases, but not both. To the best of our knowledge, this work is the first application of metamorphic test selection.

9.7 Conclusion

This chapter presents a metamorphic test selection approach which aims to reduce the cost of MT while preserving as much of its effectiveness as possible. Our approach employs a multi-objective search algorithm, for which we define the problem representation, new genetic operators, and objective functions to guide the search towards more cost-effective solutions. We define an empirical evaluation based on an industrial case study from the elevation domain, and we perform experiments to assess the effectiveness of our approach. We also provide a replication package for these experiments [AAPA22b].

The experimental results show that there is a great potential to improve the cost-effectiveness of MT, and we conclude that our approach outperforms all the baselines at this task. We discover that our approach performs best with domain-specific objective function in our experiments. Finally, we derive some lessons learned from this work, and we propose several promising research avenues that can be explored in the future.

Part V

Final Remarks

Chapter 10

Conclusion

This chapter concludes the thesis. Section 10.1 summarizes the contributions, addresses the validation of the hypotheses and highlights the limitations of the proposed solutions. Section 10.2 discusses the lessons learned from the work on this thesis. Finally, Section 10.3 highlights the potential future work related to this thesis, including the industry transfer of the solutions, their application in new domains, and further research avenues.

10.1 Summary of the Contributions

The automation of Cyber-Physical System (CPS) development is a challenging task due to fundamental issues such as the test oracle problem. This dissertation presents various approaches to achieve this automation, focusing mainly on the use of Metamorphic Testing (MT) [CCY98] to automate the verification process of systems which could not be tested automatically otherwise. The following list summarizes the specific contributions made:

1. A taxonomy for eliciting requirements of a DevOps architecture suitable for CPSs has been presented. This taxonomy was developed based on the requirements from Orona and Bombardier Transportation, which are major industrial CPS developers from the elevation and railway domains respectively. The goal of this taxonomy is to assist with the elicitation, management and reuse of the requirements by providing a framework to classify them. So far, this work comprises DevOps subsystems for deployment, monitoring, validation and integration, but extensions for additional tasks such as automated failure recovery are expected.

2. A DevOps architecture suitable for CPSs is proposed based on the taxonomy and requirements derived from the previous contribution. This architecture comprises various microservices which communicate though REST and MQTT endpoints, such that the DevOps system can be easily extended with new microservices

implementing other subsystems. The version we present includes subsystems for deployment, monitoring and verification, as well as an automation server which orchestrates all of them. The verification subsystem can incorporate the MT techniques presented in this thesis, among others.

3. A method for identifying performance-related Metamorphic Relations (MRs) based on the Performance Variation (PV) Metamorphic Relation Pattern (MRP) has been defined. MRs derived from this pattern exploit input changes that have a predictable impact on performance metrics, allowing the definition of a test oracle for systems where no other alternatives exist. Our experiments with Orona's elevator dispatcher and an open-source Autonomous Driving System (ADS) show that the MRs generated following this approach outperform threshold-based oracles and regression testing, while also not requiring to define threshold values or provide a baseline.

4. An evolutionary approach for generating MRs in a fully-automated way based on system executions labelled as correct or incorrect has been developed. The algorithm evolves a population of MRs with the goals of minimizing the false positives and false negatives, based on the provided correct and incorrect system executions. More specifically, two sub-populations are co-evolved: One prioritizing the minimization of false positives, and the other prioritizing the minimization of false negatives. Both populations exchange their best MRs regularly, and the MR with the fewest false positives, fewest false negatives, and the least complexity, in that order of priority, is selected as the best. Our experiments with Orona's case study revealed that the MRs generated with this automated approach outperform the MRs we generated manually in our previous contribution.

5. An approach to optimize MT by applying multi-objective test selection has been proposed. Due to the high cost of testing CPSs, such a technique is necessary in order to increase the cost-effectiveness of the process to the point where running the test suite repeatedly is feasible. We propose the use of the Non-dominated Sorting Genetic Algorithm II (NSGA-II), in combination with novel genetic operators and various black-box fitness functions, to perform the metamorphic test selection. The experimental evaluation on Orona's case study revealed that our approach outperforms a random baseline, and that our new genetic operators increase the efficiency of the technique.

Overall, this thesis proposed approaches to enable a fully-automated verification process for CPSs. We believe that these contributions advance the state of the art towards the point where reliance on human intervention can be removed or minimized. Since processes which rely on manual intervention do not scale, this is a necessary step to make the future development of increasingly complex CPSs sustainable.

10.1.1 Hypotheses Validation

This section discusses whether the hypotheses presented in Section 4.2 have been validated through the research performed in this thesis.

Hypothesis 1

> "MT can detect faulty behaviours of CPSs that cannot be detected with specified oracles".

We tested this hypothesis throughout Part III by generating performance MRs using two different approaches, one manual and the other automated. In Chapter 7, manually generated MRs are applied to the industrial multi-elevator system from Orona, as well as an open-source ADS, both of which lacked fully-automated specified oracles. We compared the effectiveness of MT at detecting faults with the results of regression oracles for the multi-elevator system, and with threshold-based oracles for the ADS. These baselines were the best automated test oracles we had available for these systems. Our experimental results showed that the MRs detected faults which the baseline oracles were unable to identify. On the other hand, Chapter 8 evaluated an automated method for generating MRs for CPSs, supported by a tool named GASSERTMRS. Our evaluation of the MRs generated for Orona's case study revealed that this approach outperforms our MRs generated manually for the same system. In summary, our evaluations showed that MT can detect more faulty behaviours than other existing test oracles, using either of the proposed MR generation approaches. Considering these results, we conclude that the stated hypothesis has been validated.

Hypothesis 2

> "The definition of MRPs can help define effective MRs for different domains".

Chapter 7 presents an approach for identifying MRs for CPSs based on a general MRP named Performance Variation (PV). This pattern encourages the definition of input transformations that have a predictable effect on the performance or Quality of Service (QoS) measures of the system. We employ the PV pattern to identify various Metamorphic Relation Input Patterns (MRIPs) for two different CPSs, a multi-elevator system and an ADS. Each of these MRIPs is then combined with QoS metrics specific to the Systems Under Test (SUTs) in order to instantiate concrete MRs. The empirical evaluation employed a randomly generated test suite and seeded faults in order to asses the false positives and ability to identify failures of the MRs and the other test oracles. As stated in the previous hypothesis validation, the experimental results showed that these MRs were more effective than any of the alternative automated oracles. Some false positives were observed during the experiments with some MRs, but their number was small enough to be manageable. Considering the results, we conclude that the hypothesis has been validated.

Hypothesis 3

“The use of Genetic Programming (GP) can allow the automatic generation of effective MRs”.

This hypothesis was tested by defining and implementing a tool which generates MRs automatically with GP based on labelled execution traces which show correct and incorrect behaviours of the SUT. Chapter 8 describes this approach, in which a co-evolutionary genetic algorithm evolves a population of MRs with the objectives of minimizing false positives, false negatives, and the complexity of the MRs. Our initial implementation of this approach, GASSERTMRS, was used to generate output relations for the elevator system MRIPs defined in Chapter 7 for the previous hypothesis validation. Our empirical evaluation with this case study revealed that the MRs generated by GASSERTMRS either matched or outperformed the manually generated MRs in terms of mutation score. On the other hand, the automated approach did result in a higher number of false positives, as well as more complex MRs. Nevertheless, neither of these issues was severe enough to be considered unmanageable. An extension of this approach, GENMORPH, was also developed in order to remove the need of user-provided MRIPs, thus allowing the fully-automated generation of MRs from scratch. We implemented this new tool targeting the generation of Java methods, were popular test generation and mutation testing tools could be leveraged in order to demonstrate the full automation of MR generation with only the SUT given as an input. We presented and evaluated multiple strategies for generating the input and output relations of the MRs, and we also incorporate a filtering step to identify and discard invalid MRs. We performed an experimental evaluation of GENMORPH using 10 methods from the Apache Commons Math library as SUTs. The results showed that our best strategy was capable of generating valid and effective (average mutation score >10%) MRs for 8 out of 10 methods. Considering the results from both of these empirical evaluations, we conclude that the stated hypothesis has been validated.

Hypothesis 4

“Multi-objective search algorithms can be used to increase the cost-effectiveness of MT by selecting a subset of the metamorphic tests”.

Chapter 9 from Part IV presents a search-based multi-objective metamorphic test selection approach which aims to increase the cost-effectiveness of test suites used for MT. Our approach employs the Non-dominated Sorting Genetic Algorithm II (NSGA-II) and a set of custom genetic operators specifically designed for the metamorphic test selection problem. We define a cost function and various generic and domain-specific black-box objective functions for our multi-elevator system case study. Our empirical evaluation first assesses the effectiveness of our algorithm versus Random Search (RS), concluding that NSGA-II results in a significantly more effective (~60% average improvement) test suite in terms of the ratio of total execution time versus achieved mutation score (using the Hypervolume (HV) quality indicator). We also evaluated the effectiveness of our new genetic operators, and the experiments revealed moderately better results (~3% average improvement). Finally, we compared the effectiveness of the different objective function combinations, and concluded that the domain-specific fitness functions (e.g. maximize passengers count) were more effective than generic ones (e.g. maximize input diversity). Overall, the results of applying our metamorphic test selection approach were positive, and demonstrated that 90% of the mutation score obtained with the whole test suite could be achieved for only 50% of the total cost for Orona's multi-elevator case study. Given these results, we believe that the stated hypothesis has been validated.

10.1.2 Limitations of the Proposed Solutions

This section discusses some of the limitations that the proposed solutions might have in practice.

Regarding the requirements taxonomy and proposed architecture for CPS DevOps systems, we note that they are initial proposals which address the most basic needs of the two industrial case studies. On the one hand, the DevOps subsystems we define in this work encompass only the minimum set of functionality we expect, and further work is planned to incorporate failure prediction and recovery subsystems. On the other hand, even though we developed this work using industrial CPS developers from the elevation and railway domains, other sub-domains may have requirements that have not been addressed in our work. In this regard, we employed a microservices-based architecture precisely because of its flexibility [OEC17, GL18], since we expect practitioners from various domains to extend or modify it in different ways depending on their specific requirements.

As for the rest of our contributions, we note that the MT approaches have been validated with Orona's multi-elevator system, with the open-source ADS case study also being used for evaluating our manually-generated MRs in Chapter 7. Considering this, additional evaluations need to be performed with other SUTs in order to assert that our methods are generally applicable for verifying CPSs. Regarding tool support, we note that our implementations require some work in order to adapt different SUTs to them. Future work will improve the experience of adapting these tools to support new data formats and other requirements that might arise from different CPSs.

Our proposal for generating MRs based on the PV pattern has the obvious limitation of needing to identify the MRs manually, which is a complex task which requires practical experience with the domain and the SUT [SFSRC16]. Fortunately, MRs are highly reusable, even across different product versions and configurations, so this overhead is only an issue when first adopting MT. On the other hand, we also observed that some of our MRs yielded false positives, although their number was easily manageable. Unfortunately, false positives or negatives are commonplace in performance testing, as the frontier between correct and incorrect behaviours is often unclear [SDPMBS20]. Finally, MT requires the execution of multiple test cases in order to check whether a MR holds, so this technique can be marginally more costly than using traditional test oracles which apply over individual test executions. This is a particular issue in the context of CPSs, where executing tests can be very costly [BG11], especially at the X-in-the-Loop levels which require real-time executions [SH09]. The metamorphic test selection approach we present in Chapter 9 aims to mitigate this problem.

As for our automated MR generation techniques, they overcome the limitations related to identifying the MRs manually, but also introduce new ones. On the one hand, our approach requires the collection of a representative set of system executions labelled as correct or incorrect. Ideally, the incorrect test executions should be obtained based on historical faults of the SUT, but this is often not possible due to the lack of a database documenting such faults. Artificially seeded faults can be used as an alternative, but they may not be representative of real faults, potentially leading to ineffective MRs being generated. A set of previously validated test executions or a trusted configuration of the system is also required in order to obtain the correct test executions. On the other hand, this automated approach is computationally expensive, with our configurations using time budgets of 30 minutes for each MR generated. Apart from that, the complexity of the automatically generated MRs was higher than that of manually generated ones, which generally makes them harder to interpret for the human engineers. Finally, the expressiveness of our implementation is currently limited to a type system with Boolean or numeric expressions. We plan to extend our tool to support more complex types, such as sets and lists, in the future, which will allow the generation of MRs that currently can not be expressed by our implementation.

Our metamorphic test selection approach currently only considers black box objectives. Two of these objectives (input diversity and MR coverage) are generic, but the others are domain-specific for the multi-elevator SUT. Considering that our experimental results showed that the domain-specific objectives were significantly more effective, our current approach would require the definition of new objective functions when using it on a different SUT. Thus, future work should evaluate this approach on more SUTs and derive general guidelines on how to select appropriate objectives for a system.

10.2 Lessons Learned

This section summarizes lessons learned from the research carried out during this Ph.D. thesis. These lessons can be employed as a guide by researchers and industrial practitioners.

- *Metamorphic Testing (MT) is a viable solution to alleviate the oracle problem in CPSs.* This technique enables the definition of automated test oracles for systems where no other alternative solution exists. This is particularly the case when non-functional properties such as performance metrics are involved, since defining the expected outcome for each system configuration and test input is often infeasible [STDRC17, STDRC18]. MRs are often configuration-agnostic, and therefore describe properties of the system that are highly reusable without parametrization. This technique has already been applied to CPSs from many other domains, including Wireless Sensor Networks [CCC$^+$07], autonomous drones [LPMS17], or self-driving cars [TPJR18, ZZZ$^+$18, ZS19].

- *Defining valid and effective MRs is difficult.* In this thesis we evaluate two alternative approaches, one manual and the other automated, each with its own advantages and drawbacks. The manual approach places a certain burden on the domain experts, who need to identify such MRs. On the other hand, the automated approach is computationally expensive, and still requires some effort to understand the generated MRs. We also found that, for some systems, it may not be possible to define performance MRs that never result in false alarms. In our elevation case study, for instance, the elevator dispatcher implements an algorithm which aims to optimize metrics such as the Average Waiting Time (AWT), and the MRs we defined for it assume that this algorithm is optimal. In reality, the dispatching algorithm cannot be perfectly optimal in every scenario, because it needs to make assumptions about the passengers that will arrive in the immediate future and their destinations. As a consequence, not all the sub-optimal behaviours observed in isolated scenarios indicate the presence of a fault, since attempting to fix some specific case may introduce performance regressions in other cases [NCRL15]. There is currently no workaround for this issue, and performance failures identified by any automated test oracle, including MRs, need to be reviewed manually.

- *Search-Based Software Engineering (SBSE) approaches are effective at solving CPS verification problems.* During this thesis, SBSE techniques have been used for automatic MR generation, as well as for metamorphic test selection. In both cases, we have concluded that search-based algorithms are effective at solving the complex multi-objective CPS verification problems that we have encountered. For the MR generation, a co-evolutionary algorithm proposed by Terragni et al. has been used, in which the fitness functions aim to minimize false positives and false negatives [TJTP20]. For metamorphic test selection, the more common NSGA-II algorithm is used to obtain pareto-optimal cost-effective solutions to the problem. Both evolutionary algorithms have yielded positive results for the problems where we applied them. Several research publications have already demonstrated the effectiveness of SBSE for solving CPS test generation, selection and prioritization [MNBB16, AWM+17a, ANBS18, AWM+19], as well as for generating test oracles [GGM+14, ZCH+14, ZZC+19, TJTP20, MPAF21].

- *Implementation of generic CPS verification techniques and tooling is difficult.* The heterogeneous nature of CPSs leads to developers using different tool-chains and environments to develop them. Matlab/Simulink is one of the most popular tools for developing CPSs, since it is an environment which supports modelling physical and software components [DVDBSR13, Mat22]. However, other CPS developers might use a different modelling environment, such as Modelica [FPA+20, 216]. For our case studies, the Orona's multi-elevator systems use the C programming language for software development, and the domain-specific Elevate simulator to model the physical components [Lim22]. On the other hand, the open-source ADS uses Matlab/Simulink [Mat]. Implementing tools which can integrate with any of these environments is not feasible, so most solutions are specific to an environment (e.g. Matlab/Simulink [MNBB16]). In our case, the tools developed during this thesis can be adapted for different systems, but require some manual work to adapt. Even at a more conceptual level, some approaches, such as those based white-box metrics like coverage, are not always applicable due to practical limitations (e.g. can not generate instrumented code to measure coverage).

10.3 Future Prospects

This section highlights the potential future work related to this thesis from three different perspectives: Industry transfer, application of the proposed solutions in new domains and further research avenues.

10.3.1 Industry Transfer

The research performed by the Engineering Faculty from the University of Mondragon is industry oriented. The main motivating case study we use throughout the chapters of this thesis is Orona's multi-elevator system, which is an industrial CPS deployed in many buildings throughout the world. Our collaboration with Orona has allowed us to use their software in order to evaluate our proposed solutions, which we have demonstrated that are effective in this context. The next logical step is to transfer these solutions to Orona for their eventual adoption in their development process. Many of these solutions have been or are being integrated within the Adeptness project [ade], through which they will be transferred to the industrial partners of the project: Orona (elevation) and Bombardier Transportation (railway). Still, further work on tool support needs to be done in order to encourage a wider industrial adoption of our proposed solutions.

10.3.2 Application in New Domains

The solutions proposed in this dissertation are generic for any type of CPSs, and particularly suitable for systems where performance or QoS metrics are relevant. However, most of our evaluation is centered around a single industrial system from the elevation domain, with an additional open-source ADS used in one of the chapters. In the future, our solutions will have to be evaluated in CPSs from different domains in order to assess their generalizability. We foresee that some adjustments will have to be made to our methods in order to accommodate new domains, and more general lessons will be learned from this exercise.

10.3.3 Further Research

Some of the contributions from this dissertation lead to research avenues that have not been explored yet.

Regarding the DevOps architecture we defined, future work will include defining new subsystems and evaluating them. The MT techniques developed in this thesis are also being integrated into this architecture, and will be transferred to the industry in this context through the Adeptness project [ade]. As mentioned in the limitations section, most of our contributions have been validated with either one (DevOps architecture and MT) or two (DevOps requirements taxonomy) industrial case studies, and the cost and effectiveness of our approaches in the context of CPSs from other domains still has to be investigated.

As for the approaches to define MRs, further research needs to be done on how useful the PV pattern we have proposed is for defining new MRs for other CPSs. Similar research on performance metamorphic testing [STDRC17, STDRC18] is being done on various types of systems [CCC+07, JJB+19, BBSB20], but the domain of CPSs remains largely unexplored in this regard. For our automated solution, extensions to augment the expressiveness of the tool and new strategies to increase the efficiency of the process are being actively developed and evaluated. When such extensions are defined, the increased search-space and the resulting high computational cost will be major concerns to address. Another research avenue for the automatically generated MRs would be assessing their understandability, and possibly developing methods to make these MRs simpler. This assessment should ideally involve a human study with engineers from an industrial case study, so as to evaluate how easy the generated MRs really are to interpret in a real scenario.

The empirical results we present throughout Part III represent a lower bound of the cost-effectiveness of MT, as we use randomly generated test suites in our experiments. The practical effectiveness of MT also needs to be assessed in combination with more effective test case generation, selection, or prioritization techniques [GB03, BS11, CCL+12, DGZ13], including our metamorphic test selection approach. On the topic of practical usefulness, our work does not yet address the process after a metamorphic test failure is found, i.e., fault localization [WGL+16]. Some approaches based on metamorphic slices have already been proposed [XWCX13], but their usefulness in a real-world CPS use case has not been evaluated yet.

Finally, our metamorphic test selection contribution resulted in the conclusion that domain-specific objective functions are the most effective, but no general guidelines on which type of properties should be used as objective functions were derived. Answering that question would require evaluating our approach on more case studies, until some general conclusions can be drawn on how to select good objective functions. Furthermore, the effectiveness of white-box fitness functions, such as code coverage, has also not been evaluated in our work.

Bibliographic References

[AAI+21] Aitor Arrieta, Jon Ayerdi, Miren Illarramendi, Aitor Agirre, Goiuria Sagardui, and Maite Arratibel. Using machine learning to build test oracles: an industrial case study on elevators dispatching algorithms. In *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 30–39. IEEE, 2021.

[AAPA22a] Jon Ayerdi, Aitor Arrieta, Ernest Bota Pobee, and Maite Arratibel. Multi-objective metamorphic test case selection: an industrial case study. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2022.

[AAPA22b] Jon Ayerdi, Aitor Arrieta, Ernest Bota Pobee, and Maite Arratibel. Replication package for metamorphic test selection, 2022. Last access: Sep 2022.

[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*, pages 1–10, 2011.

[ABB+21] John Ahlgren, Maria Berezin, Kinga Bojarczuk, Elena Dulskyte, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Maria Lomeli, Erik Meijer, et al. Testing web enabled simulation at scale using metamorphic testing. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 140–149. IEEE, 2021.

[ABG+20] Pekka Abrahamsson, Goetz Botterweck, Hadi Ghanbari, Martin Gilje Jaatun, Petri Kettunen, Tommi J Mikkonen, Anila Mjeda, Jürgen Münch, Anh Nguyen Duc, Barbara Russo, et al. Towards a secure devops approach for cyber-physical systems: An industrial perspective.

International Journal of Systems and Software Security and Protection (IJSSSP), 11(2):38–57, 2020.

[ABS94] Todd M Austin, Scott E Breach, and Gurindar S Sohi. Efficient detection of all pointer and array access errors. In *Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation*, pages 290–301, 1994.

[ade] Adeptness project. https://adeptness.eu. Last access: Aug 2022.

[ADH$^+$89] Hiralal Agrawal, Richard DeMillo, R_ Hathaway, William Hsu, Wynne Hsu, Edward W Krauser, Rhonda J Martin, Aditya P Mathur, and Eugene Spafford. Design of mutant operators for the c programming language. Technical report, 1989.

[AGA$^+$20] Jon Ayerdi, Aitor Garciandia, Aitor Arrieta, Wasif Afzal, Eduard Enoiu, Aitor Agirre, Goiuria Sagardui, Maite Arratibel, and Ola Sellin. Towards a taxonomy for eliciting design-operation continuum requirements of cyber-physical systems. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 280–290, 2020.

[AHH04] Konstantinos Adamopoulos, Mark Harman, and Robert M Hierons. How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution. In *Genetic and evolutionary computation conference*, pages 1338–1349. Springer, 2004.

[ALGT21] Afsoon Afzal, Claire Le Goues, and Christopher Steven Timperley. Mithra: Anomaly detection as an oracle for cyberphysical systems. *IEEE Transactions on Software Engineering*, 2021.

[Alu15] Rajeev Alur. *Principles of cyber-physical systems*. MIT press, 2015.

[AMS16] Eman Alatawi, Tim Miller, and Harald Søndergaard. Generating source inputs for metamorphic testing using dynamic symbolic execution. In *Proceedings of the 1st International Workshop on Metamorphic Testing*, pages 19–25, 2016.

[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.

[Apa22] Apache. Apache Commons Math. https://commons.apache.org/proper/commons-math/, 2022.

[ARF+18] Muhammad Alam, Joao Rufino, Joaquim Ferreira, Syed Hassan Ahmed, Nadir Shah, and Yuanfang Chen. Orchestration of microservices for iot using docker and edge computing. *IEEE Communications Magazine*, 56(9):118–123, 2018.

[Arr22a] Aitor Arrieta. Multi-objective metamorphic follow-up test case selection for deep learning systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1327–1335, 2022.

[Arr22b] Aitor Arrieta. On the cost-effectiveness of composite metamorphic relations for testing deep learning systems. In *2022 IEEE/ACM 7th International Workshop on Metamorphic Testing (MET)*, pages 42–47. IEEE, 2022.

[ASA+20] Jon Ayerdi, Sergio Segura, Aitor Arrieta, Goiuria Sagardui, and Maite Arratibel. Qos-aware metamorphic testing: An elevation case study. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 104–114. IEEE, 2020.

[ATA+21] Jon Ayerdi, Valerio Terragni, Aitor Arrieta, Paolo Tonella, Goiuria Sagardui, and Maite Arratibel. Generating metamorphic relations for cyber-physical systems with genetic programming: An industrial case study. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, pages 1264–1274, New York, NY, USA, 2021. Association for Computing Machinery.

[ATA+22] Jon Ayerdi, Valerio Terragni, Aitor Arrieta, Paolo Tonella, Goiuria Sagardui, and Maite Arratibel. Evolutionary generation of metamorphic relations for cyber-physical systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 15–16, 2022.

[ATF09] Wasif Afzal, Richard Torkar, and Robert Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[AVAS22] Aitor Arrieta, Pablo Valle, Joseba A Agirre, and Goiuria Sagardui. Some seeds are strong: Seeding strategies for search-based test case selection. *ACM Transactions on Software Engineering and Methodology*, 2022.

[Avi95] Algirdas Avizienis. The methodology of n-version programming. *Software fault tolerance*, 3:23–46, 1995.

[AVS+22a] Jon Ayerdi, Pablo Valle, Sergio Segura, Aitor Arrieta, Goiuria Sagardui, and Maite Arratibel. Performance-driven metamorphic testing of cyber-physical systems. *IEEE Transactions on Reliability*, pages 1–19, 2022.

[AVS+22b] Jon Ayerdi, Pablo Valle, Sergio Segura, Aitor Arrieta, Goiuria Sagardui, and Maite Arratibel. Replication package for the autonomous driving system. https://github.com/pablovalle/MT-AutonomousVehicle, 2022. Last access: Sep 2022.

[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)*, pages 688–697. IEEE, 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 and 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*, pages 297–306, 2016.

[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*, pages 1053–1060, 2016.

[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.

[AY07] Andrea Arcuri and Xin Yao. Coevolving programs and unit tests from their specification. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 397–400, 2007.

[AY08] Andrea Arcuri and Xin Yao. A novel co-evolutionary approach to automatic software bug fixing. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 162–168. IEEE, 2008.

[AY14] Andrea Arcuri and Xin Yao. Co-evolutionary automatic programming for software development. *Information Sciences*, 259:412–432, 2014.

[B$^+$12] Nguyen Thanh Binh et al. Mutation operators for simulink models. In *2012 Fourth International Conference on Knowledge and Systems Engineering*, pages 54–59. IEEE, 2012.

[BAS15] Gina Barney and Lutfi Al-Sharif. *Elevator traffic handbook: theory and practice*. Routledge, 2015.

[BB07] Markus F Brameier and Wolfgang Banzhaf. A comparison with tree-based genetic programming. *Linear Genetic Programming*, pages 173–192, 2007.

[BBSB20] Mohamed Boussaa, Olivier Barais, Gerson Sunyé, and Benoit Baudry. Leveraging metamorphic testing to automatically detect inconsistencies in code generator families. *Software Testing, Verification and Reliability*, 30(1):e1721, 2020.

[BCK$^+$16] Arlinta Christy Barus, Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, and Heinz W Schmidt. The impact of source test case selection on the effectiveness of metamorphic testing. In *2016 IEEE/ACM 1st International Workshop on Metamorphic Testing (MET)*, pages 5–11. IEEE, 2016.

[Ber07] Antonia Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, pages 85–103. IEEE Computer Society, 2007.

[BFFR18] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In *Lectures on Runtime Verification*, pages 1–33. Springer, 2018.

[BFHN18] Alexey Bakhirkin, Thomas Ferrère, Thomas A Henzinger, and Dejan Ničković. The first-order logic of signals: keynote. In *Proceedings of the International Conference on Embedded Software*, page 1. IEEE Press, 2018.

[BG11] Radhakisan Baheti and Helen Gill. Cyber-physical systems. *The impact of control technology*, 12(1):161–166, 2011.

[BGE+21] Arianna Blasi, Alessandra Gorla, Michael D Ernst, Mauro Pezzè, and Antonio Carzaniga. Memo: Automatically identifying metamorphic relations in javadoc comments for test automation. *Journal of Systems and Software*, 181:111041, 2021.

[BGK+22] Christian Birchler, Nicolas Ganz, Sajad Khatiri, Alessio Gambi, and Sebastiano Panichella. Cost-effective simulation-based test selection in self-driving cars software with sdc-scissor. In *29th IEEE International Conference on Software Analysis, Evolution, and Reengineering, Honolulu, USA (online), 15-18 March 2022*. ZHAW Zürcher Hochschule für Angewandte Wissenschaften, 2022.

[BGT16] Björn Butzin, Frank Golatowski, and Dirk Timmermann. Microservices approach for the internet of things. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–6. IEEE, 2016.

[BHM+14] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2014.

[BJB+21] Chaima Boufaied, Maris Jukss, Domenico Bianculli, Lionel Claude Briand, and Yago Isasi Parache. Signal-based properties of cyber-physical systems: Taxonomy and logic-based characterization. *Journal of Systems and Software*, 174:110881, 2021.

[BKH+15] Yi Bian, Serkan Kirbas, Mark Harman, Yue Jia, and Zheng Li. Regression test case prioritisation for guava. In *International Symposium on Search Based Software Engineering*, pages 221–227. Springer, 2015.

[BLC13] Lionel Briand, Yvan Labiche, and Kathy Chen. A multi-objective genetic algorithm to rank state-based test cases. In *International Symposium on Search Based Software Engineering*, pages 66–80. Springer, 2013.

[BMB+20] Chaima Boufaied, Claudio Menghi, Domenico Bianculli, Lionel Briand, and Yago Isasi Parache. Trace-checking signal-based temporal properties: A model-driven approach. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1004–1015. IEEE, 2020.

[Bör10] Egon Börger. The abstract state machines method for high-level system design and analysis. In *Formal Methods: State of the Art and New Directions*, pages 79–116. Springer, 2010.

[BS11] Gagandeep Batra and Jyotsna Sengupta. An efficient metamorphic testing technique using genetic algorithm. In *International Conference on Information Intelligence, Systems, Technology and Management*, pages 180–188. Springer, 2011.

[BS19] Anu Bajaj and Om Prakash Sangwan. A systematic literature review of test case prioritization using genetic algorithms. *IEEE Access*, 7:126355–126375, 2019.

[BSB+16] Sreram Balasubramaniyan, Seshadhri Srinivasan, Furio Buonopane, B Subathra, Jüri Vain, and Srini Ramaswamy. Design and verification of cyber-physical systems using truetime, evolutionary optimization and uppaal. *Microprocessors and microsystems*, 42:37–48, 2016.

[BSH17] Halil Beglerovic, Michael Stolz, and Martin Horn. Testing of autonomous vehicles using surrogate models and stochastic optimization. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2017.

[CCC+07] Wing Kwong Chan, Tsong Y Chen, Shing Chi Cheung, TH Tse, and Zhenyu Zhang. Towards the testing of power-aware software applications for wireless sensor networks. In *International Conference on Reliable Software Technologies*, pages 84–99. Springer, 2007.

[CCL$^+$12] Leilei Chen, Lizhi Cai, Jiang Liu, Zhenyu Liu, Shiyan Wei, and Pan Liu. An optimized method for generating cases of metamorphic testing. In *2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012)*, pages 439–443. IEEE, 2012.

[CCY98] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. Metamorphic testing: a new approach for generating next test cases. Technical report, Technical Report HKUST-CS98-01, Department of Computer Science, The Hong Kong University of Science and Technology, 1998.

[CEN01] CENELEC. 50128: Railway Application: Communications, Signaling and Processing Systems, Software For Railway Control and Protection Systems. In *Standard Official Document*. European Committee for Electrotechnical Standardization, 2001.

[CES86] Edmund M. Clarke, E Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[CGT$^+$20] Yiqun T Chen, Rahul Gopinath, Anita Tadakamalla, Michael D Ernst, Reid Holmes, Gordon Fraser, Paul Ammann, and René Just. Revisiting the relationship between fault detection, test adequacy criteria, and test set size. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 237–249, 2020.

[CHTZ04] Tsong Yueh Chen, DH Huang, TH Tse, and Zhi Quan Zhou. Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004)*, pages 569–583. Polytechnic University of Madrid, 2004.

[CJKO01] David W Corne, Nick R Jerram, Joshua D Knowles, and Martin J Oates. Pesa-ii: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd annual conference on genetic and evolutionary computation*, pages 283–290, 2001.

[CKL16] Younghwan Cho, Jeongho Kim, and Eunseok Lee. History-based test case prioritization for failure information. In *2016 23rd Asia-Pacific*

Software Engineering Conference (APSEC), pages 385–388. IEEE, 2016.

[CKL+18] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)*, 51(1):1–27, 2018.

[CKLT04] Tsong Yueh Chen, Fei-Ching Kuo, Ying Liu, and Antony Tang. Metamorphic testing and testing with special values. In *SNPD*, pages 128–134, 2004.

[CLH+16] Henry Coles, Thomas Laurent, Christopher Henard, Mike Papadakis, and Anthony Ventresque. Pit: a practical mutation testing tool for java. In *Proceedings of the 25th international symposium on software testing and analysis*, pages 449–452, 2016.

[CLM04] Tsong Yueh Chen, Hing Leung, and Ieng Kei Mak. Adaptive random testing. In *Annual Asian Computing Science Conference*, pages 320–329. Springer, 2004.

[CP13] Fedor Chernogorov and Jani Puttonen. User satisfaction classification for minimization of drive tests qos verification. In *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 2165–2169. IEEE, 2013.

[CPS18] Yuqi Chen, Christopher M Poskitt, and Jun Sun. Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 648–660. IEEE, 2018.

[CPX16] Tsong Yueh Chen, Pak-Lok Poon, and Xiaoyuan Xie. Metric: Metamorphic relation identification based on the category-choice framework. *Journal of Systems and Software*, 116:177–190, 2016.

[CW19] Benoit Combemale and Manuel Wimmer. Towards a model-based devops for cyber-physical systems. In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 84–94. Springer, 2019.

[CZ11] Edmund M Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 1–12. Springer, 2011.

[CZC13] Yuxiang Cao, Zhi Quan Zhou, and Tsong Yueh Chen. On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions. In *2013 13th International Conference on Quality Software*, pages 153–162. IEEE, 2013.

[DAS$^+$18] Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M Rao, RP Bose, Neville Dubash, and Sanjay Podder. Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 118–128. ACM, 2018.

[Deb14] Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer, 2014.

[DELT17] Alastair F Donaldson, Hugues Evrard, Andrei Lascu, and Paul Thomson. Automated testing of graphics shader compilers. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–29, 2017.

[DGZ13] Guowei Dong, Tao Guo, and Puhan Zhang. Security assurance with program path analysis and metamorphic testing. In *2013 IEEE 4th International Conference on Software Engineering and Service Science*, pages 193–197. IEEE, 2013.

[DJ13] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4):577–601, 2013.

[DKW08] Vijay D'silva, Daniel Kroening, and Georg Weissenbacher. A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1165–1178, 2008.

[DL16] Alastair F Donaldson and Andrei Lascu. Metamorphic testing for (graphics) compilers. In *Proceedings of the 1st international workshop on metamorphic testing*, pages 44–47, 2016.

[DPAM02] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[DRML22] Elizabeth Dinella, Gabriel Ryan, Todd Mytkowicz, and Shuvendu K Lahiri. Toga: A neural method for test oracle generation. 2022.

[DSPBA13] Luciano S De Souza, Ricardo BC Prudêncio, Flavia de A Barros, and Eduardo H da S Aranha. Search based constrained test case selection using execution effort. *Expert systems with applications*, 40(12):4887–4896, 2013.

[DTPRK22] Alejandra Duque-Torres, Dietmar Pfahl, Rudolf Ramler, and Claus Klammer. A replication study on predicting metamorphic relations at unit testing level. In *2022 IEEE 29th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 698–708. IEEE, 2022.

[DVDBSR13] Yanja Dajsuren, Mark GJ Van Den Brand, Alexander Serebrenik, and Serguei Roubtsov. Simulink models are also software: Modularity assessment. In *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pages 99–106, 2013.

[DW81] Martin D Davis and Elaine J Weyuker. Pseudo-oracles for non-testable programs. In *Proceedings of the ACM'81 Conference*, pages 254–257, 1981.

[dWOJ+07] Eckert Claudia M de Weck Olivier, Clarkson P John, et al. A classification of uncertainty for early product and system design. *Guidelines for a Decision Support Method Adapted to NPD Processes*, pages 159–160, 2007.

[DZZ+22] Yao Deng, Xi Zheng, Tianyi Zhang, Huai Liu, Guannan Lou, Miryung Kim, and Tsong Yueh Chen. A declarative metamorphic testing framework for autonomous driving. *IEEE Transactions on Software Engineering*, 2022.

[EGHS16] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *Ieee Software*, 33(3):94–100, 2016.

[EPG+07] Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. The daikon system for dynamic detection of likely invariants. *Science of computer programming*, 69(1-3):35–45, 2007.

[FA11] Gordon Fraser and Andrea Arcuri. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 416–419, 2011.

[FA12] Gordon Fraser and Andrea Arcuri. Whole test suite generation. *IEEE Transactions on Software Engineering*, 39(2):276–291, 2012.

[Fel98] Robert Feldt. Generating diverse software versions with genetic programming: an experimental study. *IEE Proceedings-Software*, 145(6):228–236, 1998.

[FJA$^+$15] King Chun Foo, Zhen Ming Jiang, Bram Adams, Ahmed E Hassan, Ying Zou, and Parminder Flora. An industrial case study on the automated detection of performance regressions in heterogeneous environments. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 159–168. IEEE, 2015.

[FL09] John Fitzgerald and Peter Gorm Larsen. *Modelling systems: practical tools and techniques in software development*. Cambridge University Press, 2009.

[FNS$^+$19] Nicolas Ferry, Phu Nguyen, Hui Song, Pierre-Emmanuel Novac, Stéphane Lavirotte, Jean-Yves Tigli, and Arnor Solberg. Genesis: Continuous orchestration and deployment of smart iot systems. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 870–875. IEEE, 2019.

[FPA$^+$20] Peter Fritzson, Adrian Pop, Karim Abdelhak, Adeel Ashgar, Bernhard Bachmann, Willi Braun, Daniel Bouskela, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Rüdiger Franke, Dag Fritzson, Mahder Gebremedhin, Andreas Heuermann, Bernt Lie, Alachew Mengist, Lars Mikelsons, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Vitalij Ruge, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, John Tinnerholm, and Per Östlund. The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Modeling, Identification and Control*, 41(4):241–295, 2020.

[FPTO11] Xin Feng, David Lorge Parnas, TH Tse, and Tony O'Callaghan. A comparison of tabular expression-based testing strategies. *IEEE Transactions on Software Engineering*, 37(5):616–634, 2011.

[GAA+21a] Aitor Gartziandia, Aitor Arrieta, Aitor Agirre, Goiuria Sagardui, and Maite Arratibel. Using regression learners to predict performance problems on software updates: a case study on elevators dispatching algorithms. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 135–144, 2021.

[GAA+21b] Aitor Gartziandia, Jon Ayerdi, Aitor Arrieta, Shaukat Ali, Tao Yue, Aitor Agirre, Goiuria Sagardui, and Maite Arratibel. Microservices for continuous deployment, monitoring and validation in cyber-physical systems: an industrial case study for elevators systems. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pages 46–53. IEEE, 2021.

[GAA+22] Aitor Gartziandia, Aitor Arrieta, Jon Ayerdi, Miren Illarramendi, Aitor Agirre, Goiuria Sagardui, and Maite Arratibel. Machine learning-based test oracles for performance testing of cyber-physical systems: An industrial case study on elevators dispatching algorithms. *Journal of Software: Evolution and Process*, 2022.

[GB03] Arnaud Gotlieb and Bernard Botella. Automated metamorphic testing. In *Proceedings 27th Annual International Computer Software and Applications Conference. COMPAC 2003*, pages 34–40. IEEE, 2003.

[GBHS10] Zhongxian Gu, Earl T Barr, David J Hamilton, and Zhendong Su. Has the bug really been fixed? In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 55–64. IEEE, 2010.

[GGM+14] Alberto Goffi, Alessandra Gorla, Andrea Mattavelli, Mauro Pezzè, and Paolo Tonella. Search-based synthesis of equivalent method sequences. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 366–376. ACM, 2014.

[GL18] Javad Ghofrani and Daniel Lübke. Challenges of microservices architecture: A survey on the state of the practice. *ZEUS*, 2018:1–8, 2018.

[GM07] Ralph Guderlei and Johannes Mayer. Statistical metamorphic testing testing programs with random output by means of statistical hypothesis tests and metamorphic testing. In *Seventh International Conference on Quality Software (QSIC 2007)*, pages 404–409. IEEE, 2007.

[GMSF07] Ralph Guderlei, Johannes Mayer, Christoph Schneckenburger, and Frank Fleischer. Testing randomized software by means of statistical hypothesis tests. In *Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting*, pages 46–54, 2007.

[Gol89] David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989(102):36, 1989.

[GR09] Ashutosh Gupta and Andrey Rybalchenko. Invgen: An efficient invariant generator. In *International Conference on Computer Aided Verification*, pages 634–640. Springer, 2009.

[GRH16] Gregory Gay, Sanjai Rayadurgam, and Mats PE Heimdahl. Automated steering of model-based test oracles to admit real program behaviors. *IEEE Transactions on Software Engineering*, 43(6):531–555, 2016.

[GS12] Edward Glaessgen and David Stargel. The digital twin paradigm for future nasa and us air force vehicles. In *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*, page 1818, 2012.

[GTB$^+$17] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: State of the art. *arXiv preprint arXiv:1702.00686*, 2017.

[GV17] Michael Grieves and John Vickers. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In *Transdisciplinary perspectives on complex systems*, pages 85–113. Springer, 2017.

[HA08] Mark Harman and Nadia Alshahwan. Automated session data repair for web application regression testing. In *2008 1st International Conference on Software Testing, Verification, and Validation*, pages 298–307. IEEE, 2008.

[HAB13] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(1):1–42, 2013.

[Har11] Mark Harman. Making the case for morto: Multi objective regression test optimization. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 111–114. IEEE, 2011.

[HC10] Alan Hevner and Samir Chatterjee. Design science research in information systems. In *Design research in information systems*, pages 9–22. Springer, 2010.

[HCL19] Yang He, Jincai Chen, and Ping Lu. Testing dynamic composition of semantic internet of things services based on qos. *IEEE Access*, 7:113103–113113, 2019.

[Hil90] W Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42(1-3):228–234, 1990.

[HJ01] Mark Harman and Bryan F Jones. Search-based software engineering. *Information and software Technology*, 43(14):833–839, 2001.

[HLN14] M Mahmudul Hasan, Pericles Loucopoulos, and Mara Nikolaidou. Classification and qualitative analysis of non-functional requirements approaches. In *Enterprise, Business-Process and Information Systems Modeling*, pages 348–362. Springer, 2014.

[HMPR04] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[HMSY10] Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza, and Shin Yoo. Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical software engineering and verification*, pages 1–59. Springer, 2010.

[HMSY13] Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. A comprehensive survey of trends in oracles for software testing. *University of Sheffield, Department of Computer Science, Tech. Rep. CS-13-01*, 2013.

[HMZ12] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, 45(1):1–61, 2012.

[Hol75] John H Holland. Adaptation in natural and artificial systems, 1975.

[HPH+16] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. Comparing white-box and black-box test prioritization. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 523–534. IEEE, 2016.

[HRS94] KJ Hughes, RM Rankin, and CT Sennett. Taxonomy for requirements analysis. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 176–179. IEEE, 1994.

[HZZ+15] Dan Hao, Lu Zhang, Lei Zang, Yanbo Wang, Xingxia Wu, and Tao Xie. To be optimal or not in test-case prioritization. *IEEE Transactions on Software Engineering*, 42(5):490–505, 2015.

[Ikea] Ikerlan. Adeptness: Annex a requirements and validation tests. https://adeptness.eu/wp-content/uploads/2020/09/D1.1-ANNEX-A-Requirements-and-validation-tests.pdf. Last access: Aug 2022.

[Ikeb] Ikerlan. Adeptness: Requirements and ethics. https://adeptness.eu/wp-content/uploads/2020/11/D1.1-REQUIREMENTS_v1.1.pdf. Last access: Aug 2022.

[JBR+93] Matthias Jarke, Janis Bubenko, Colette Rolland, Allistair Sutcliffe, and Y Vassilou. Theories underlying requirements engineering: An overview of nature at genesis. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 19–31. IEEE, 1993.

[JCHT16] Gunel Jahangirova, David Clark, Mark Harman, and Paolo Tonella. Test oracle assessment and improvement. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 247–258, 2016.

[JCHT18] Gunel Jahangirova, David Clark, Mark Harman, and Paolo Tonella. Oasis: Oracle assessment and improvement tool. In *Proceedings of* *the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 368–371, 2018.

[JCHT19] Gunel Jahangirova, David Clark, Mark Harman, and Paolo Tonella. An empirical validation of oracle improvement. *IEEE Transactions on Software Engineering*, 47(8):1708–1728, 2019.

[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.

[JH15] Zhen Ming Jiang and Ahmed E Hassan. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, 41(11):1091–1118, 2015.

[Jin11] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[JJB$^+$19] Owen Johnston, Darryl Jarman, Jeffrey Berry, Zhi Quan Zhou, and Tsong Yueh Chen. Metamorphic relations for detection of performance anomalies. In *2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET)*, pages 63–69. IEEE, 2019.

[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*, pages 654–665, 2014.

[JST21] Gunel Jahangirova, Andrea Stocco, and Paolo Tonella. Quality metrics and oracles for autonomous vehicles testing. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 194–204. IEEE, 2021.

[Jus14] René Just. The major mutation framework: Efficient and scalable mutation analysis for java. In *Proceedings of the 2014 international symposium on software testing and analysis*, pages 433–436, 2014.

[KB13] Upulee Kanewala and James M Bieman. Using machine learning techniques to detect metamorphic relations for programs without test oracles. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 1–10. IEEE, 2013.

[KBBH16] Upulee Kanewala, James M Bieman, and Asa Ben-Hur. Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. *Software testing, verification and reliability*, 26(3):245–269, 2016.

[KEH$^+$09] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.

[Kel76] Robert M Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[KFK14] Aaron Kane, Thomas Fuhrman, and Philip Koopman. Monitor based oracles for cyber-physical system testing: Practical experience report. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 148–155. IEEE, 2014.

[KGJV83] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[KIJT18] Muhammad Khatibsyarbini, Mohd Adham Isa, Dayang NA Jawawi, and Rooster Tumeng. Test case prioritization approaches in regression testing: A systematic literature review. *Information and Software Technology*, 93:74–93, 2018.

[KJMG17] Rafaqut Kazmi, Dayang NA Jawawi, Radziah Mohamad, and Imran Ghani. Effective regression test case selection: A systematic literature review. *ACM Computing Surveys (CSUR)*, 50(2):1–32, 2017.

[KLT16] Hui Kang, Michael Le, and Shu Tao. Container and microservice driven design for cloud infrastructure devops. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 202–211. IEEE, 2016.

[KM14] Siddhartha Kumar Khaitan and James D McCalley. Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal*, 9(2):350–365, 2014.

[KM18] Hojat Khosrowjerdi and Karl Meinke. Learning-based testing for autonomous systems using spatial and temporal requirements. In *Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis*, pages 6–15, 2018.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

[Koz94] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[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, 2002.

[LAS14] Vu Le, Mehrdad Afshari, and Zhendong Su. Compiler validation via equivalence modulo inputs. *ACM Sigplan Notices*, 49(6):216–226, 2014.

[LBR06] Gary T Leavens, Albert L Baker, and Clyde Ruby. Preliminary design of jml: A behavioral interface specification language for java. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–38, 2006.

[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.

[Lee08] Edward A Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*, pages 363–369. IEEE, 2008.

[LFN+17] Remo Lachmann, Michael Felderer, Manuel Nieke, Sandro Schulze, Christoph Seidl, and Ina Schaefer. Multi-objective black-box test case selection for system testing. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1311–1318, 2017.

[LGÁW15] Mikael Lindvall, Dharmalingam Ganesan, Ragnar Árdal, and Robert E Wiegand. Metamorphic model-based testing applied on nasa dat–an experience report. In *2015 IEEE/ACM 37th IEEE International*

Conference on Software Engineering, volume 2, pages 129–138. IEEE, 2015.

[LHH07] Zheng Li, Mark Harman, and Robert M Hierons. Search algorithms for regression test case prioritization. *IEEE Transactions on software engineering*, 33(4):225–237, 2007.

[LHWM17] Duo Lu, Dijiang Huang, Andrew Walenstein, and Deep Medhi. A secure microservice framework for iot. In *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 9–18. IEEE, 2017.

[Lim09] Milind G Limaye. *Software testing*. Tata McGraw-Hill Education, 2009.

[Lim22] Peters Research Limited. Elevate. https://peters-research.com, 2022. Last access: Sep 2022.

[LJAZ21] Johannes Lampel, Sascha Just, Sven Apel, and Andreas Zeller. When life gives you oranges: detecting and diagnosing intermittent job failures at mozilla. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1381–1392, 2021.

[LKC19] Pericles Loucopoulos, Evangelia Kavakli, and Natalia Chechina. Requirements engineering for cyber physical production systems. In *International Conference on Advanced Information Systems Engineering*, pages 276–291. Springer, 2019.

[LKF20] Stephan Lukasczyk, Florian Kroiß, and Gordon Fraser. Automated unit test generation for python. In *International Symposium on Search Based Software Engineering*, pages 9–24. Springer, 2020.

[LKM22] Pericles Loucopoulos, Evangelia Kavakli, and Julien Mascolo. Requirements engineering for cyber physical production systems: The e-core approach and its application. *Information Systems*, 104:101677, 2022.

[LKTC13] Huai Liu, Fei-Ching Kuo, Dave Towey, and Tsong Yueh Chen. How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering*, 40(1):4–22, 2013.

[LLC12] Huai Liu, Xuan Liu, and Tsong Yueh Chen. A new method for constructing metamorphic relations. In *2012 12th International Conference on Quality Software*, pages 59–68. IEEE, 2012.

[LMLK14] Richard Lai, S Mahmood, R Lai, and Y S Kim. Survey of component-based software development. *The Institution of Engineering and Technology*, 3(May 2007):58–64, 2014.

[LNB$^+$17] Bing Liu, Shiva Nejati, Lionel C Briand, et al. Improving fault localization for simulink models using search-based testing and prediction models. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 359–370. IEEE, 2017.

[LO16] Nan Li and Jeff Offutt. Test oracle strategies for model-based testing. *IEEE Transactions on Software Engineering*, 43(4):372–395, 2016.

[LPMS17] Mikael Lindvall, Adam Porter, Gudjon Magnusson, and Christoph Schulze. Metamorphic model-based testing of autonomous systems. In *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*, pages 35–41. IEEE, 2017.

[LRK$^+$19] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM Computing Surveys (CSUR)*, 52(6):1–35, 2019.

[LS09] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The journal of logic and algebraic programming*, 78(5):293–303, 2009.

[LS16] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press, 2016.

[Mat] MathWorks Student Competitions Team". mathworks/vehicle-pure-pursuit. Last access: Aug 2022.

[Mat22] MathWorks. Matlab/simulink. https://www.mathworks.com/products/simulink.html, 2022. Last access: Aug 2022.

[MB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[MB19] Manish Motwani and Yuriy Brun. Automatically generating precise oracles from structured natural language specifications. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 188–199. IEEE, 2019.

[MdA22] Facundo Molina, Marcelo d'Amorim, and Nazareno Aguirre. Fuzzing class specifications. 2022.

[MFS90] Barton P Miller, Louis Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. *Communications of the ACM*, 33(12):32–44, 1990.

[MG06] Johannes Mayer and Ralph Guderlei. An empirical study on the selection of good metamorphic relations. In *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, volume 1, pages 475–484. IEEE, 2006.

[MGS$^+$17] Morten Mossige, Arnaud Gotlieb, Helge Spieker, Hein Meling, and Mats Carlsson. Time-aware test case execution scheduling for cyber-physical systems. In *International Conference on Principles and Practice of Constraint Programming*, pages 387–404. Springer, 2017.

[MHD15] Debajyoti Mondal, Hadi Hemmati, and Stephane Durocher. Exploring test suite diversification and code coverage in multi-objective test case selection. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10. IEEE, 2015.

[MHH13] Haroon Malik, Hadi Hemmati, and Ahmed E Hassan. Automatic detection of performance deviations in the load testing of large scale systems. In *2013 35th international conference on software engineering (ICSE)*, pages 1012–1021. IEEE, 2013.

[MLM18] Elena Markoska and Sanja Lazarova-Molnar. Towards smart buildings performance testing as a service. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 277–282. IEEE, 2018.

[MMGG95] Brad L. Miller, Brad L. Miller, David E. Goldberg, and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3):193–212, 1995.

[MMT18] Gianluca Mezzetti, Anders Møller, and Martin Toldam Torp. Type regression testing to detect breaking changes in node. js libraries. In *32nd European Conference on Object-Oriented Programming (ECOOP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[MN04] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[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*, pages 595–606, 2016.

[MNBB18] Reza Matinnejad, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. Test generation and test prioritization for simulink models with dynamic behavior. *IEEE Transactions on Software Engineering*, 45(9):919–944, 2018.

[MNGB19] Claudio Menghi, Shiva Nejati, Khouloud Gaaloul, and Lionel C Briand. Generating automated and online test oracles for simulink models with continuous and uncertain behaviors. In *Proceedings of the 2019 27th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 27–38, 2019.

[MPAF21] Facundo Molina, Pablo Ponzio, Nazareno Aguirre, and Marcelo Frias. Evospex: An evolutionary algorithm for learning postconditions. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1223–1235. IEEE, 2021.

[MRB19] K Monisha and M Rajasekhara Babu. A novel framework for healthcare monitoring system through cyber-physical system. In *Internet of things and personalized healthcare systems*, pages 21–36. Springer, 2019.

[MS76] Webb Miller and David L. Spooner. Automatic generation of floating-point test data. *IEEE Transactions on Software Engineering*, (3):223–226, 1976.

[MS11] Karl Meinke and Muddassar A Sindhu. Incremental learning-based testing for reactive systems. In *International Conference on Tests and Proofs*, pages 134–151. Springer, 2011.

[MSK09a] Christian Murphy, Kuang Shen, and Gail Kaiser. Automatic system testing of programs without test oracles. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 189–200. ACM, 2009.

[MSK09b] Christian Murphy, Kuang Shen, and Gail Kaiser. Using jml runtime assertion checking to automate metamorphic testing in applications without test oracles. In *2009 International Conference on Software Testing Verification and Validation*, pages 436–445. IEEE, 2009.

[MT17] Marco Miglierina and Damian A Tamburri. Towards omnia: A monitoring factory for quality-aware devops. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, pages 145–150, 2017.

[NCRL15] Adrian Nistor, Po-Chun Chang, Cosmin Radoi, and Shan Lu. Caramel: Detecting and fixing performance problems that have non-intrusive fixes. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 902–912. IEEE, 2015.

[ND12] Srinivas Nidhra and Jagruthi Dondeti. Black box and white box testing techniques-a literature review. *International Journal of Embedded Systems and Applications (IJESA)*, 2(2):29–50, 2012.

[NFE$^+$19] Phu Hong Nguyen, Nicolas Ferry, Gencer Erdogan, Hui Song, Stéphane Lavirotte, Jean-Yves Tigli, and Arnor Solberg. A systematic mapping study of deployment and orchestration approaches for iot. *IoTBDS*, pages 69–82, 2019.

[NJT13] Adrian Nistor, Tian Jiang, and Lin Tan. Discovering, reporting, and fixing performance bugs. In *2013 10th working conference on mining software repositories (MSR)*, pages 237–246. IEEE, 2013.

[NKF94] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on software engineering*, 20(10):760–773, 1994.

[OEC17] Rory V O'Connor, Peter Elger, and Paul M Clarke. Continuous software engineering-a microservices architecture perspective. *Journal of Software: Evolution and Process*, 29(11):e1866, 2017.

[OP21] Mitchell Olsthoorn and Annibale Panichella. Multi-objective test case selection through linkage learning-based crossover. In *International Symposium on Search Based Software Engineering*, pages 87–102. Springer, 2021.

[Oro22] Orona. Orona group. https://www.orona-group.com/, 2022. Last access: Sep 2022.

[216] Open Source Modelica Consortium (OSMC). Openmodelica. https://www.openmodelica.org/, 2022. Last access: Aug 2022.

[Par10] David Lorge Parnas. Really rethinking. *Computer*, (1):28–34, 2010.

[Par11] David Lorge Parnas. Precise documentation: The key to better software. In *The Future of Software Engineering*, pages 125–148. Springer, 2011.

[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 International Symposium on Software Testing and Analysis*, pages 354–365, 2016.

[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 *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 936–946. IEEE, 2015.

[PKT17] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering*, 44(2):122–158, 2017.

[PLEB07] Carlos Pacheco, Shuvendu K Lahiri, Michael D Ernst, and Thomas Ball. Feedback-directed random test generation. In *29th International Conference on Software Engineering (ICSE'07)*, pages 75–84. IEEE, 2007.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

[PODPDL14] Annibale Panichella, Rocco Oliveto, Massimiliano Di Penta, and Andrea De Lucia. Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering*, 41(4):358–383, 2014.

[PP98] Dennis K Peters and David Lorge Parnas. Using test oracles generated from program documentation. *IEEE Transactions on Software Engineering*, 24(3):161–173, 1998.

[PT20] Nenad Petrovic and Milorad Tosic. Smada-fog: Semantic model driven approach to deployment and adaptivity in fog computing. *Simulation Modelling Practice and Theory*, 101:102033, 2020.

[PTP$^+$19] Konstantinos Papachristou, Traianos Theodorou, Stavros Papadopoulos, Aikaterini Protogerou, Anastasios Drosou, and Dimitrios Tzovaras. Runtime and routing security policy verification for enhanced quality of service of iot networks. In *2019 Global IoT Summit (GIoTS)*, pages 1–6. IEEE, 2019.

[PWAY16] Dipesh Pradhan, Shuai Wang, Shaukat Ali, and Tao Yue. Search-based cost-effective test case selection within a time budget: An empirical study. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 1085–1092, 2016.

[PZ14] Mauro Pezze and Cheng Zhang. Automated test oracles: A survey. In *Advances in computers*, volume 95, pages 1–48. Elsevier, 2014.

[Ral18] Paul Ralph. Toward methodological guidelines for process theories and taxonomies in software engineering. *IEEE Transactions on Software Engineering*, 45(7):712–735, 2018.

[Ram00] Ganesan Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Transactions on Programming languages and Systems (TOPLAS)*, 22(2):416–430, 2000.

[RH17] Jennifer Rowley and Richard Hartley. *Organizing knowledge: an introduction to managing access to information*. Routledge, 2017.

[RKC$^+$06] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, Jeff Skowronek, and Linda Devine. Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and cohen'sd indices the most appropriate choices. In *annual meeting of the Southern Association for Institutional Research*, pages 1–51. Citeseer, 2006.

[RKS$^+$16] Hassan Reza, Christoffer Korvald, Jeremy Straub, Justin Hubber, Nicholas Alexander, and Abhinav Chawla. Toward requirements engineering of cyber-physical systems: Modeling cubesat. In *2016 IEEE Aerospace Conference*, pages 1–13. IEEE, 2016.

[Rom85] G-C Roman. A taxonomy of current issues in requirements engineering. *Computer*, 18(04):14–23, 1985.

[RUCH99] Gregg Rothermel, Roland H Untch, Chengyun Chu, and Mary Jean Harrold. Test case prioritization: An empirical study. In *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*, pages 179–188. IEEE, 1999.

[RVT$^+$18] Luis F Rivera, Norha M Villegas, Gabriel Tamura, Miguel Jiménez, and Hausi A Müller. Uml-driven automated software deployment. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, pages 257–268, 2018.

[SA92] J Michael Spivey and JR Abrial. *The Z notation*. Prentice Hall Hemel Hempstead, 1992.

[Sch02] Rolf Schwitter. English as a formal specification language. In *Proceedings. 13th International Workshop on Database and Expert Systems Applications*, pages 228–232. IEEE, 2002.

[SDPMBS20] Ana B. Sánchez, Pedro Delgado-Pérez, Inmaculada Medina-Bulo, and Sergio Segura. Tandem: A taxonomy and a dataset of real-world performance bugs. *IEEE Access*, 8:107214–107228, 06 2020.

[SDTC17] Sergio Segura, Amador Durán, Javier Troya, and Antonio Ruiz Cortés. A template-based approach to describing metamorphic relations. In *Proceedings of the 2nd International Workshop on Metamorphic Testing*, pages 3–9. IEEE Press, 2017.

[SDTRC19] Sergio Segura, Amador Durán, Javier Troya, and Antonio Ruiz-Cortés. Metamorphic relation patterns for query-based systems. In *2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET)*, pages 24–31. IEEE, 2019.

[SFP$^+$19] Chang-Ai Sun, An Fu, Pak-Lok Poon, Xiaoyuan Xie, Huai Liu, and Tsong Yueh Chen. Metric+: A metamorphic relation identification technique based on input plus output domains. *IEEE Transactions on Software Engineering*, 47(9):1764–1785, 2019.

[SFSRC16] Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. A survey on metamorphic testing. *IEEE Transactions on software engineering*, 42(9):805–824, 2016.

[SG06] Bart Selman and Carla P Gomes. Hill-climbing search. *Encyclopedia of cognitive science*, 81:82, 2006.

[SG20] Helge Spieker and Arnaud Gotlieb. Adaptive metamorphic testing with contextual bandits. *Journal of Systems and Software*, 165:110574, 2020.

[SH09] Hesham Shokry and Mike Hinchey. Model-based verification of embedded software. 2009.

[SHBRC11] Sergio Segura, Robert M Hierons, David Benavides, and Antonio Ruiz-Cortés. Automated metamorphic testing on the analyses of feature models. *Information and Software Technology*, 53(3):245–258, 2011.

[SHD16] Hema Srikanth, Charitha Hettiarachchi, and Hyunsook Do. Requirements based test prioritization using risk factors: An industrial study. *Information and Software Technology*, 69:71–83, 2016.

[SHNF15] Weiyi Shang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. Automated detection of performance regressions using regression models on clustered performance counters. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 15–26, 2015.

[SK18] Prashanta Saha and Upulee Kanewala. Fault detection effectiveness of source test case generation strategies for metamorphic testing. In *Proceedings of the 3rd International Workshop on Metamorphic Testing*, pages 2–9, 2018.

[SK22] Madhusudan Srinivasan and Upulee Kanewala. Metamorphic relation prioritization for effective regression testing. *Journal of Software: Testing, Verification and Reliability*, 2022.

[SKIH11] Seyed Reza Shahamiri, Wan Mohd Nasir Wan Kadir, Suhaimi Ibrahim, and Siti Zaiton Mohd Hashim. An automated framework for software test oracle. *Information and Software Technology*, 53(7):774–788, 2011.

[SLF$^+$22] Chang-ai Sun, Baoli Liu, An Fu, Yiqiang Liu, and Huai Liu. Path-directed source test case generation and prioritization in metamorphic testing. *Journal of Systems and Software*, 183:111091, 2022.

[SMT22] SMTLIB. The satisfiability modulo theories libraries. http://smtlib.cs.uiowa.edu/, 2022.

[SNdT22] Andrea Stocco, Paulo J Nunes, Marcelo d'Amorim, and Paolo Tonella. Thirdeye: Attention maps for safe autonomous driving systems. In *Proceedings of 37th IEEE/ACM International Conference on Automated Software Engineering, ASE*, volume 22, 2022.

[SPE$^+$07] Sara Sprenkle, Lori Pollock, Holly Esquivel, Barbara Hazelwood, and Stacey Ecott. Automated oracle comparators for testing web applications. In *The 18th IEEE International Symposium on Software Reliability (ISSRE'07)*, pages 117–126. IEEE, 2007.

[SPTRC18] Sergio Segura, José A Parejo, Javier Troya, and Antonio Ruiz-Cortés. Metamorphic testing of restful web apis. In *Proceedings of the 40th International Conference on Software Engineering*, pages 882–882, 2018.

[SSSG14] Narayanamurthy Srinivas, Panditi Stefan Schmidt, and Ralf Garrelfs. Mil/sil/pil approach a new paradigm in model based development, 2014.

[STDRC17] Sergio Segura, Javier Troya, Amador Durán, and Antonio Ruiz-Cortés. Performance metamorphic testing: Motivation and challenges. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*, pages 7–10. IEEE, 2017.

[STDRC18] Sergio Segura, Javier Troya, Amador Durán, and Antonio Ruiz-Cortés. Performance metamorphic testing: A proof of concept. *Information and Software Technology*, 98:1–4, 2018.

[Sto11] David R Stoutemyer. Ten commandments for good default expression simplification, 2011.

[STZC20] Sergio Segura, Dave Towey, Zhi Quan Zhou, and Tsong Yueh Chen. Metamorphic testing: Testing the untestable. *IEEE Software*, 37(3):46–53, 2020.

[SW65] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[SWCT20] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. Misbehaviour prediction for autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 359–371, 2020.

[SWKIH12] Seyed Reza Shahamiri, Wan Wan-Kadir, Suhaimi Ibrahim, and Siti Zaiton Mohd Hashim. Artificial neural networks as multi-networks automated test oracle. *Automated Software Engineering*, 19(3):303–334, 2012.

[SWYS11] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. A survey of cyber-physical systems. In *2011 international conference on wireless communications and signal processing (WCSP)*, pages 1–6. IEEE, 2011.

[TDUE07] Tetsuo Tomiyama, Valentina DÆAmelio, Jill Urbanic, and Waguih ElMaraghy. Complexity of multi-disciplinary design. *CIRP annals*, 56(1):185–188, 2007.

[TJP$^+$20] Valerio Terragni, Gunel Jahangirova, Mauro Pezze, Paolo Tonella, et al. Technical report: Tr-precrime-2020-02. 2020.

[TJTP20] Valerio Terragni, Gunel Jahangirova, Paolo Tonella, and Mauro Pezzè. Evolutionary improvement of assertion oracles. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1178–1189, 2020.

[TJTP21] Valerio Terragni, Gunel Jahangirova, Paolo Tonella, and Mauro Pezzè. Gassert: a fully automated tool to improve assertion oracles. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 85–88. IEEE, 2021.

[TPJR18] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314. ACM, 2018.

[TVS18] Kleanthis Thramboulidis, Danai C Vachtsevanou, and Alexandros Solanos. Cyber-physical microservices: An iot-based framework for manufacturing systems. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 232–239. IEEE, 2018.

[UBBM17] Muhammad Usman, Ricardo Britto, Jürgen Börstler, and Emilia Mendes. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. *Information and Software Technology*, 85:43–59, 2017.

[UPL06] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing. 2006.

[UQES20] Miriam Ugarte Querejeta, Leire Etxeberria, and Goiuria Sagardui. Towards a devops approach in cyber physical production systems using digital twins. In *International Conference on Computer Safety, Reliability, and Security*, pages 205–216. Springer, 2020.

[Val21] Pablo Valle. Metamorphic testing of autonomous vehicles: A case study on simulink. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 105–107. IEEE, 2021.

[VCHB$^+$18] Michael Vierhauser, Jane Cleland-Huang, Sean Bayley, Thomas Krismayer, Rick Rabiser, and Pau Grünbacher. Monitoring cps at runtime-a case study in the uav domain. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 73–80. IEEE, 2018.

[VD00] András Vargha and Harold D Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong.

Journal of Educational and Behavioral Statistics, 25(2):101–132, 2000.

[VK15] Vijay K Vaishnavi and William Kuechler. *Design science research methods and patterns: innovating information and communication technology*. Crc Press, 2015.

[WAG13] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1493–1500, 2013.

[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, 2016.

[WE95] Stephanie White and Michael Edwards. A requirements taxonomy for specifying complex systems. In *Proceedings of First IEEE International Conference on Engineering of Complex Computer Systems. ICECCS'95*, pages 373–376. IEEE, 1995.

[Wey82] Elaine J Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[WGL$^+$16] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016.

[WHHT16] Stefan Wiesner, Jannicke Baalsrud Hauge, Florian Haase, and Klaus-Dieter Thoben. Supporting the requirements elicitation process for cyber-physical product-service systems through a gamified approach. In *IFIP International Conference on Advances in Production Management Systems*, pages 687–694. Springer, 2016.

[Whi94] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[WLS20] Muhammad Waseem, Peng Liang, and Mojtaba Shahin. A systematic mapping study on microservices architecture in devops. *Journal of Systems and Software*, 170:110798, 2020.

[Woo07] Robert F Woolson. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*, pages 1–3, 2007.

[WTZC18] Zhenyu Wang, Dave Towey, Zhi Quan Zhou, and Tsong Yueh Chen. Metamorphic testing for adobe analytics data collection javascript library. In *Proceedings of the 3rd International Workshop on Metamorphic Testing*, pages 34–37, 2018.

[XGKS14] Zhiwei Xu, Kehan Gao, Taghi M Khoshgoftaar, and Naeem Seliya. System regression test planning with a fuzzy expert system. *Information Sciences*, 259:532–543, 2014.

[XHM$^+$11] Xiaoyuan Xie, Joshua WK Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84(4):544–558, 2011.

[XWCX13] Xiaoyuan Xie, W Eric Wong, Tsong Yueh Chen, and Baowen Xu. Metamorphic slice: An application in spectrum-based fault localization. *Information and Software Technology*, 55(5):866–879, 2013.

[XZC$^+$20] Xiaoyuan Xie, Zhiyi Zhang, Tsong Yueh Chen, Yang Liu, Pak-Lok Poon, and Baowen Xu. Mettle: A metamorphic testing approach to assessing and validating unsupervised machine learning systems. *IEEE Transactions on Reliability*, 69(4):1293–1322, 2020.

[YH07] Shin Yoo and Mark Harman. Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 international symposium on Software testing and analysis*, pages 140–150, 2007.

[YH10] Shin Yoo and Mark Harman. Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *Journal of Systems and Software*, 83(4):689–701, 2010.

[YH12] Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability*, 22(2):67–120, 2012.

[YL12] Yuen Tak Yu and Man Fai Lau. Fault-based test suite prioritization for specification-based testing. *Information and Software Technology*, 54(2):179–202, 2012.

[ZAY19a] Man Zhang, Shaukat Ali, and Tao Yue. Uncertainty-wise test case generation and minimization for cyber-physical systems. *Journal of Systems and Software*, 153:1–21, 2019.

[ZAY$^+$19b] Man Zhang, Shaukat Ali, Tao Yue, Roland Norgren, and Oscar Okariz. Uncertainty-wise cyber-physical system test modeling. *Software & Systems Modeling*, 18(2):1379–1418, 2019.

[ZCH$^+$14] Jie Zhang, Junjie Chen, Dan Hao, Yingfei Xiong, Bing Xie, Lu Zhang, and Hong Mei. Search-based inference of polynomial metamorphic relations. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 701–712. ACM, 2014.

[ZJKK15] Xi Zheng, Christine Julien, Miryung Kim, and Sarfraz Khurshid. Perceptions on the state of the art in verification and validation in cyber-physical systems. *IEEE Systems Journal*, 11(4):2614–2627, 2015.

[ZL07] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, 103, 2001.

[ZS19] Zhi Quan Zhou and Liqun Sun. Metamorphic testing of driverless cars. *Communications of the ACM*, 62(3):61–67, 2019.

[ZSA$^+$16] Man Zhang, Bran Selic, Shaukat Ali, Tao Yue, Oscar Okariz, and Roland Norgren. Understanding uncertainty in cyber-physical systems: a conceptual model. In *European conference on modelling foundations and applications*, pages 247–264. Springer, 2016.

[ZSCT18] Zhi Quan Zhou, Liqun Sun, Tsong Yueh Chen, and Dave Towey. Metamorphic relations for enhancing system understanding and use. *IEEE Transactions on Software Engineering*, 46(10):1120–1154, 2018.

[ZZC$^+$19] Bo Zhang, Hongyu Zhang, Junjie Chen, Dan Hao, and Pablo Moscato. Automatic discovery and cleansing of numerical metamorphic relations. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 235–245. IEEE, 2019.

[ZZPL17] Pengcheng Zhang, Xuewu Zhou, Patrizio Pelliccione, and Hareton Leung. Rbf-mlmr: A multi-label metamorphic relation prediction approach using rbf neural network. *IEEE access*, 5:21791–21805, 2017.

[ZZZ$^+$18] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 132–142. IEEE, 2018.