Uncertainty-wise Testing of Cyber-Physical Systems

Shaukat Ali, Hong Lu, Shuai Wang, Tao Yue, Man Zhang

(The authors are alphabetically ordered.)

Abstract

As compared with classical software/system testing, uncertainty-wise testing explicitly addresses known uncertainty about the behavior of a System Under Test (SUT), its operating environment, and interactions between the SUT and its operational environment, across all testing phases, including test design, test generation, test optimization, and test execution, with the aim to mainly achieve the following two goals. First, uncertainty-wise testing aims to ensure that the SUT deals with known uncertainty adequately. Second, uncertainty-wise testing should be also capable of learning new (previously unknown) uncertainties such that the SUT's implementation can be improved to guard against newly learned uncertainties during its operation. The necessity to integrate uncertainty in testing is becoming imperative because of the emergence of new types of intelligent and communicating software-based systems such as Cyber-Physical Systems (CPSs). Intrinsically, such systems are exposed to uncertainty because of their interactions with highly indeterminate physical environments. In this chapter, we provide our understanding and experience of uncertainty-wise testing from the aspects of uncertainty-wise model-based testing, uncertainty-wise modeling and evolution of test ready models, and uncertainty-wise multi-objective test optimization, in the context of testing CPSs under uncertainty. Furthermore, we present our vision about this new testing paradigm and its plausible future research directions.

Keywords— Uncertainty-wise Testing; Cyber-Physical System; Belief Test Ready Model; Model Evolution; Model-Based Testing.

1 INTRODUCTION

Uncertainty is unpreventable in the behavior of a Cyber-Physical Systems (CPSs) given its close interaction with its physical environment [1-4]. Predicting the exact behavior of the physical environment of a CPS is not viable, and a common practice is to make assumptions about the physical environment during the design and testing of the CPS. The correct behavior of a CPS is only guaranteed when such assumptions prove to be true. Given the complexity of problems being solved by CPSs in critical domains, these systems must function safely even when experiencing uncertainty in their physical environment to avert any harms. Thus, we argue that uncertainty (i.e., lack of knowledge) in the behavior of a CPS, its operating environment, and in their interactions, must be considered explicitly during the testing phase. To this end, we propose, in this chapter, *Uncertainty-wise Testing (UWT)*, which is a new testing paradigm that explicitly takes uncertainty into consideration at various testing phases, including test design, test generation, test optimization, and test execution, to make sure that the implementation of a CPS is sufficiently robust against its uncertain physical environment.

This chapter introduces *UWT* particularly in the context of CPSs. Note that this chapter surveys the existing works in this area rather than providing new scientific and research contributions; therefore, we refer to relevant references where further details can be consulted. In particular, we survey the following three key topics in the field of *UWT*.

First, we introduce *Uncertainty-wise Model-Based Testing (UWMBT)* that focuses on model-based testing of CPSs in the presence of uncertainty with the explicit consideration of known uncertainty on test ready models. Such test ready models capture the behavior of a CPS together with uncertainty in its environment and interactions between them. Particularly, we will introduce uncertainty-wise test modeling and evolution, which aims to improve the quality of modeled test ready models with uncertainty using the real operational data of CPSs. With machine learning techniques, invariants and observed uncertainties can be abstracted from such data that can be used to further enhance test ready models. Such improved test ready models can be used to generate additional test cases as compared to the initial test ready models. In addition, uncertainty-wise test generation will also be discussed. Notice that such test ready models are the key inputs for an uncertainty-wise model-based test generation tool that generates test cases from the models using uncertainty-wise test case generation strategies. The generated test cases are then executed on a CPS to ensure that it handles the known uncertainty specified in the models during its operation, in addition to discovering unknown uncertainty, i.e., the uncertainty not specified in the models.

Second, we will survey about uncertainty-wise multi-objective test optimization. Since test case execution on a real CPS is not only time consuming, but also resource intensive, test optimization is necessary. When performing multi-objective test optimization, in addition to cost and effectiveness objectives uncertainty-wise optimization objectives must also be considered. In this topic, we particularly focus on uncertainty-wise test set minimization and uncertainty-wise test case prioritization. Finally, we also provide a detailed survey of existing uncertainty-wise testing techniques for CPSs.

We organize our chapter as follows. Section 2 discusses uncertainty-wise model-based testing and uncertainty-wise test modeling and evolution in detail followed by uncertainty-wise multi-objective test optimization (Section 3). Section 4 discusses the detailed state-of-the-art related to CPS testing under uncertainty. Finally, Section 5 summarizes and concludes the chapter. To make the paper more readable, we provide a detailed list of abbreviations that will be used in this chapter as shown in Table 1.

Table 1. List of Abbreviations

Abbreviation	Description
APML	All Paths with Maximum Length
ASP	All Simple Paths
BTRM	Belief Test Ready Model
CPS	Cyber-Physical System
ETSI	European Telecommunications Standards Institute
ICT	Information and Communications Technology
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
ITU	International Telecommunication Union
JCGM	Joint Committee for Guides in Metrology
MARTE	Modeling And Analysis of Real-Time Embedded Systems
MBE	Model-Based Engineering
MBT	Model-Based Testing
MDE	Model Driven Engineering
NIST	The National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
OCL	Object Constraint Language
OMA	Open Mobile Alliance
OMG	Object Management Group
OSI	Open Systems Interconnection
RFP	Request For Proposal
RTF	Revision Task Force
SACM	Structured Assurance Case Metamodel
SBT	Search-Based Testing
TDL	Test Description Language
TRM	Test Ready Model
UMF	Uncertainty Modeling Framework
UML	Unified Modeling Language
UTF	Uncertainty Testing Framework
UTP	UML Testing Profile
UUP	UML Uncertainty Profile
UWMBT	Uncertainty-Wise Model-Based Testing
UWT	Uncertainty-Wise Testing
UWTG	Uncertainty-Wise Test Generation
UWTM	Uncertainty-Wise Multi-Objective Test Set Minimization
UWTP	Uncertainty-Wise Multi-Objective Test Case Prioritization

2 UNCERTAINTY-WISE MODEL-BASED TESTING

In this section, we first briefly summarize two main research trends of test generation and point readers to existing literature reviews or surveys for details in Section 2.1. Second, we discuss our perspective and vision of uncertainty-wise test generation, with a particular focus on uncertainty-wise MBT (Section 2.2). Last, we highlight two important aspects (i.e., tool support in Section 2.3 and standardization in Section 2.4) in terms of enhancing the possibility of being applied in practical and industrial settings.

2.1 Overview of Test Generation

Automated test generation [5-8] is in principle about automatically generating test cases by following certain generation strategies, including automatically generating test data, based on one or more software artifacts such as program structures and software/system design models. Depending on which type(s) of software artifacts to use, there exist many test generation techniques such as symbolic execution [9, 10], program structural coverage based test generation [11, 12], and model-based test case generation [13]. Depending on which mechanism to use for test generation, there exist other test generation techniques such as combinatorial testing [6] and search-based testing [7, 14].

When particularly looking into the research stream of MBT [15], there are many MBT techniques that have been proposed. Such techniques take very diverse types of models as input and generate test cases either automatically or semi-automatically. We term such models as Test Ready Models (TRMs) in general, which refer to all types of models that are used as input for generating test cases. TRMs can be very different, depending on testing contexts and objectives. For example, in [16], the authors proposed a restricted natural language based test case specification language (named as RTCM) to specify test case specifications (one type of TRMs), from which executable test cases can be generated. TRMs are also often specified as UML models. For example, the authors of [17] proposed a model-based framework, named as TRUST, for automatically generating executable test cases from UML state machines.

Search-based testing (SBT) [7, 8] is getting more and more attention these days. SBT is about using meta-heuristic optimization search techniques (e.g., Genetic Algorithm [18]) for enabling fully-automated or partially automated testing tasks such as generating test cases or test data. Researchers and practitioners have advanced this field by proposing varying types of testing techniques, summarized in [7, 14]. When talking about search-based test generation particularly, interesting solutions have been proposed in the last ten years. For example, EvoSuite [19] is an automated test suite generation tool for testing Java programs.

2.2 Uncertainty-wise Test Generation (UWTG)

Considering the fact that, models are the key artifacts of any MBT technique, in this section, we first put our focus on modeling methodologies. Second, we discuss uncertainty-wise test generation strategies.

2.2.1 Uncertainty-wise modeling

In this section, we first define TRMs in the context of MBT, based on which we further define TRMs with uncertainty information explicitly captured (Section 2.2.1.1). After that, we discuss uncertainty modeling from a general perspective (Section 2.2.1.2). In the last three subsections, we discuss modeling notations and methodologies for modeling TRMs with uncertainty (Section 2.2.1.3), how they fit into a typical system development lifecycle (Section 2.2.1.4), and what measures one should take to ensure the quality of developed TRMs with uncertainty via verification and model evolution techniques (Section 2.2.1.5).

2.2.1.1 Belief test ready models

We define TRMs, in the context of MBT, as models, which can come in diverse forms (e.g., UML) capturing information that is necessary and sufficient to enable MBT, for the purpose of generating test cases and other required information such as test stimuli and expected test results. TRMs are test ready in the sense that they are amenable for test generation. TRMs are models in the sense that they capture required information at an abstract level. It is, therefore, important to differentiate TRMs from models specifying test cases themselves.

TRMs are often particularly designed for the purpose of testing by test engineers, rather than system engineers. To be able to develop TRMs, test engineers need to have a certain level of understanding of system behaviors and interactions with their operating environments. In addition, as a modeler, a test engineer needs to establish required modeling skills to be able to use proposed modeling notations (e.g., UML) and a corresponding tool implementation.

Belief Test Ready Models (BTRMs) are a specific type of TRMs with uncertainty information explicitly captured as part of the models. Such uncertainty information is specified by test engineers to capture their beliefs about model elements of a BTRM, associated with one or more uncertainties due to "lack of knowledge" about a BTRM when the BTRM was developed. *Belief* and *Uncertainty* should be measured/quantified such that an MBT technique can benefit from the quantified belief and uncertainty information by proposing uncertainty-wise test generation and optimization techniques. In order to do so, a modeling methodology particularly designed for specifying and quantifying uncertainty is needed. Such a methodology ideally should be easy to use, sufficiently expressive in terms of capturing all required information, and ideally based on standard modeling notations, as we described above. Most importantly, *uncertainty modeling* is the key to developing BTRMs and for enabling *UWTG* required to capture uncertainty information explicitly.

2.2.1.2 Uncertainty modeling in general

Uncertainty Modeling in software engineering is a new modeling paradigm; therefore, new modeling notations, tools, and methodologies are required to enable the specification and modeling of uncertainty and its related concepts for the purpose of analyzing any uncertainty-related aspects of a system, and generating other artifacts (e.g., test cases) by taking the uncertainty information into account. Such an uncertainty-modeling paradigm ideally should support the specification/modeling of uncertainty at different levels of abstractions, depending on particular needs. For example, at the requirements engineering phase, uncertainty should be explicitly captured as an important attribute of requirements, similar to their other attributes such as *Verifiability* and *Completeness*. This is however not the current practice as the community lacks understanding of the importance of explicitly specifying not only requirements but also associated uncertainties and confidence of requirement engineers for the specified requirements. Another example is that uncertainty can be specified as part of TRMs, which are collectively named as BTRMs as we discussed above.

To understand what uncertainty is in the context of software engineering in general, recently, a conceptual model, named as U-Model [20], has been proposed to define uncertainty, belief, belief agent, indeterminacy source and other related concepts. U-Model was proposed from the perspective of software engineers (e.g., requirements engineers, test engineers) to understand uncertainty. In other words, U-Model was defined from a *subjective* perspective. U-Model can be considered as the first attempt towards the direction of pursuing a common understanding of uncertainty in the software engineering community. There are a few possibilities of applying U-Model. First, it can be extended for particular purposes. For example, U-Model can be extended for understanding and classifying uncertainty in Cyber-Physical Systems (CPSs). Second, U-Model can be implemented as an independent domain specific language. Third, U-Model can be integrated with other modeling notations. For instance, a UML uncertainty profile, named as UUP, has been developed, based on U-Model, to facilitate the development of BTRMs, as discussed in [21]. Another example is to integrate U-Model with other purpose-specific modeling/specification methodologies, such as use case modeling. In [22, 23], the authors presented a framework, named as U-RUCM, for the purpose of capturing uncertainties in use case models and facilitating the discovery of unknown uncertainties. It is worth mentioning that U-RUCM was built on a well-established use case modeling paradigm (Zen-RUCM) [24-26]. In that case, U-Model was used as the basis to develop, for instance, uncertain alternative flows and uncertain sentences. Among all of them, the key motivation of U-Model was to facilitate the common understanding of uncertainty in the software engineering community. As long as the common understanding is built, we believe uncertainty modeling will form a new modeling paradigm. Consequently, tools and corresponding standards will be developed and proposed such that we can put uncertainty modeling into practice.

Considering the fact SysML has been widely used for system modeling [27, 28], in the context of uncertainty-wise testing of CPSs, it is therefore also important to seek opportunities of bringing uncertainty modeling to SysML. The latest version of SysML 1.4 includes a very limited capability of modeling distribution (via e.g., «Normal», «Uniform»), which is, however, from the perspective of being "complete", and when comparing with U-Model at the conceptual level, there is a huge potential to extend this capability by 1) introducing more comprehensive list of probability distribution types (e.g., triangular distribution) and other types of measures (e.g., fuzziness and ambiguity) and 2) introducing other U-Model concepts (e.g., belief, indeterminacy source) to SysML. Another possibility is to extend the requirements modeling part of SysML by introducing subjective uncertainty to textual requirement statements. The idea is similar to U-RUCM, which however particularly focuses on use case modeling, not only on textual requirement statements.

In summary, we believe U-Model can be a starting point for developing a standardized uncertainty-modeling paradigm such that it can be integrated with other modeling solutions for different purposes in various contexts. OMG has started the standardization activities of Uncertainty Modeling [29].

2.2.1.3 Modeling belief test ready models

As we discussed earlier, BTRMs are the key input to enable MBT of a system under uncertainty. Therefore, practically useful and meaningful modeling notations, tools and methodologies are required to enable the development of such BTRMs. In [21], the authors have developed a UML-based, uncertainty-wise modeling framework, named as *UncerTum*. *UncerTum* was proposed to enable the development of BTRMs for the purpose of testing CPSs under uncertainty. In other words, it is not a generic modeling solution for developing general BTRMs. However, some aspects of *UncerTum* can serve for general BTRM modeling. Nevertheless, we are not aware of any other uncertainty modeling solutions for developing BTRMs.

UncerTum is built on a set of well-known modeling technologies, including UML, MARTE, OCL and UTP. More specifically, in the current implementation of *UncerTum*, UML class diagrams and state machines are the artifacts, on which uncertainty information is attached. The rationale behind selecting these two particular UML notations is because they are commonly used for enabling MBT [15, 30-32]. The core of *UncerTum* is the UML Uncertainty Profile (UUP), as we discussed above, which is built on U-Model. UUP enables the modeling of belief, uncertainty, and measurement and it also defines an extensive list of model libraries for specifying uncertainty patterns (e.g., *Periodic*), measures (e.g., *Probability*) and time (borrowed from MARTE). *UncerTum* has been integrated with an MBT framework named as *UncerTest* [33], which will be discussed in details in Section 2.2.2. One can adopt, adapt or extend *UncerTum* for other UML modeling notations such as UML sequence and activity diagrams in case such notations are needed to support particular MBT techniques such as [34, 35]. One can also adopt, adapt or extend *UncerTum* for developing BTRMs for testing other types of systems, in addition to CPSs. Other well-applied modeling notations (e.g., SysML) can be also extended to enable uncertainty modeling; therefore, *UncerTum* is a very nice example to exemplify how such an extension can be developed.

2.2.1.4 Modeling uncertainty in a lifecycle

Conforming to classical software/system development life cycles and following the transformation principles of MDE, there is a possibility to develop a full uncertainty modeling/tooling/methodology chain. Such a chain of uncertainty modeling can start from specifying uncertainty requirements, to modeling uncertainty at the architecture and design level, and all the way for developing BTRMs. Developing such a toolchain is very useful. For example, at the requirements engineering phase, uncertainty requirements are specified by requirements engineers, who explicitly indicate, by using certain uncertainty requirements specification methodology (e.g., U-RUCM), "places" where they lack confidence about a requirement statement and their confidence level indicating to what extent they lack knowledge on what they are specifying. The ultimate objective is to eventually eliminate explicitly specified uncertainties at the requirements engineering phase as much as possible, which is however in practice not always possible. Instead, one might want to discover as many uncertainty requirements as possible and as early as possible, such that mitigation plans can be made before it gets too late. Therefore, automated uncertainty requirements analyses are preferred in such context, as briefly discussed in [22]. For uncertainty requirements that have to be carried on to the next phase of the development lifecycle, software/system designers and developers then need to keep in mind such uncertainty requirements and find ways to mitigate them if possible. During the testing phase, uncertainty requirements can be transferred as part of BTRMs such that systems can be tested against those uncertainty requirements.

2.2.1.5 Verifying and evolving belief test ready models

TRMs are expected to be correct and complete in terms of enabling MBT before they are used as the input of an MBT solution. Therefore, it is necessary and important to verify their correctness and completeness, both syntactically and semantically. Nowadays, most of the existing modeling tools can facilitate automated checking of syntactic correctness of models. However, dedicated techniques and tools are needed to check their semantics and their completeness. Especially when verifying BTRMs, correctness and completeness of the uncertainty information of a BTRM have to be verified. Therefore, novel techniques for verifying uncertainty aspects of BTRMs or other belief models (e.g., uncertainty requirements specified in U-RUCM) are urgently needed. However, currently, we are not aware of such solutions and future development in this research direction is welcomed.

As we discussed earlier, a BTRM model captures *subjective* uncertainty at the beginning as they were specified from the perspective of one or more modelers, who are defined as belief agents in U-Model. However, there is a possibility to continuously evolve BTRMs when more and more evidence is collected to update belief agents' understanding of uncertainty, which eventually and ideally should be reflected in future revisions of original BTRMs. There might be many ways to achieve this objective. Here, in this section, to inspire readers, we introduce one of such methodologies, which is named as *UncerTolve* [36, 37]. *UncerTolve* was developed in the context of supporting MBT of CPSs under uncertainty, with the objective of evolving BTRMs continuously, motivated by the hypothesis that high-quality BTRMs have higher chance to result in high-quality test cases. In [36, 37], the authors proposed a framework for interactively evolving BTRMs, specified with *UncerTum* (Section 2.2.1). An original BTRM means a BTRM developed by a test engineer with her/his beliefs containing *subjective* uncertainties captured as part of the model. *UncerTolve* aims to mine *objective* uncertainties from real data of the system, which can be collected in various ways, including obtaining real operational data from previous deployments of the same system and/or the same types of systems and obtaining test logs from previously executed test cases. Such data that reflect execution results of the systems, are objective and can be used for deriving meaningful information that can be used to improve the quality of the BTRM. For example, previously-unknown uncertainties can be discovered from such data; the original BTRM can be refactored with more precisely defined state invariants (specified as OCL constraints, if the BTRM was developed with *UncerTum*, i.e., extended UML state machines); the uncertainty measurements of the original BTRM can be improved by reflecting not just subjective aspects, but also objective aspects. *UncerTolve* relies on machine learning techniques to achieve these objectives and we believe there exist many other techniques for enabling the evolutions of BTRMs. Future investigations are definitely needed towards this research direction.

2.2.2 *Uncertainty-wise test generation*

In this section, we discuss *UncerTest* and our visions beyond *UncerTest*, including interesting future research directions.

2.2.2.1 *UncerTest and beyond*

As we discussed earlier, test generation includes two parts: test case generation and test data generation. In the context of MBT, TRMs are used as the basis to generate both test data and test cases. Various strategies have been proposed in the literature. In the following section, we discuss potentials of using uncertainty information to guide the generation of test cases and test data.

Typical test generation in the context of MBT with TRMs specified mainly as state machines rely on structural coverage criteria such as state coverage, transition coverage, and path coverage [11, 12, 38, 39], which are defined on TRMs. For example, in *UncerTest*, BTRMs are queried to generate abstract test cases from BTRMs by following two commonly used test case generation strategies: all simple paths (ASP) and all paths with maximum length (APML). Of course, *UncerTest* can be integrated with other generation strategies and further investigation is required to understand which strategies are better in which situations.

In the context of *UWTG*, there are in principle two kinds of mechanisms to generate abstract test cases. One mechanism is about generating abstract test cases from BTRMs as other MBT techniques do, but during the generation, each test case is attached with uncertainty information (e.g., the number of uncertainties covered by the test case). One of the challenges of MBT is that an MBT technique often generates a large number of test cases; therefore, finding a way to reduce the number of test cases to be eventually executed is very critical. Hence, uncertainty information attached to each test case can be used to define test optimization heuristics/strategies, as we will discuss in Section 3. In other words, this test case generation strategy itself is not specific to *UWTG*, but deriving/calculating uncertainty information for test cases by taking uncertainty information specified in BTRMs as the input is uncertainty-wise. It is also important to mention that different theories (e.g., probability theory [40] and uncertainty theory [41]) can be applied during the uncertainty information generation process for test cases. Which theory to apply depends on how the uncertainty information was obtained and specified as part of BTRMs at the first place, which determines whether prerequisites of a specific theory are satisfied. For example, applying probability theory requires that sufficient data observed from previous executions of the SUT such that uncertainty can be measured with probability. Otherwise, one might consider using uncertainty theory [41].

The second mechanism utilizes uncertainty information specified as part of BTRMs to guide the generation of the test case. In other words, one needs to propose uncertainty-wise test generation strategies. Such a strategy can be designed, for example, to find a minimum number of paths but covering as many uncertainties as possible, with the ultimate objective of minimizing the number of test cases to be executed eventually. Notice that the difference with the first mechanism is that test optimization (e.g., test selection) is integrated as part of test case generation. The second mechanism defines two sequential steps: generation first and then optimization.

It is hard to say, in a general context, which mechanism is better at the moment, as we, the community, need to collect more experience about *UWTG*, particularly about diverse application contexts of *UWTG*, uncertainty-wise test generation, optimization strategies, and accompanied empirical studies.

Test data generation [5, 42] is also an important part of test generation. For generating executable test cases, new test data (often named as artificial data) can be generated by following some strategies or actual data that has been taken from previous operations. Sometimes, it is also possible to combine artificial data and actual data. Cost-effectively generating effective test data is always a challenge in testing. Therefore, in the literature, many test data strategies have been proposed, including random test data generations [42], boundary-value analysis based test data generations [42] and equivalent partitioning based test data generations [43]. Search algorithms have been also successfully used in the past for test data generation, as reported in [7, 8, 44]. There also exist constraint solvers that help to generate valid test data. One of such constraint solver is EsOCL [45], which was built based on search algorithms to efficiently find valid test data. In the current implementation of *UncerTest*, a random test data generation strategy was applied to generate test data for generated abstract test cases. However, in the future, it is worth investigating uncertainty-wise test data generation strategies such that high quality and uncertainty-oriented test data can be generated. For example, uncertainty related results of test executions (e.g., the number of observed uncertainties) can be used for future test data generation.

2.3 Tool Support

Automation is all about tool support. Therefore, it is important to recognize the importance of tooling in the context of test generation. There exist a large number of test generation tools. Well-known open source test generation tools include [19, 46-48] and commercial test generation tools include [49-51].

Tooling is especially important for MBT, as applying modeling tools to produce TRMs is not always straightforward. Therefore, the usability, applicability, and expressiveness of a modeling tool should be taken into account when producing an MBT framework. From industrial practitioners' perspective, it is important to select a tool with good applicability, usability, and expressiveness for easier adoption. The success of any automated MBT solution depends on these properties of its tool implementation, based on which a methodology is often provided to further ease the process of applying an MBT solution, especially considering such a solution is often non-trivial.

2.4 Standardization

Considering the fact that standards play increasingly important role in industrial practices, we conducted a survey to understand whether there are standardization bodies and standards that are relevant for uncertainty modeling. Motivating by this, we identify standardization bodies such as European Telecommunications Standards Institute

(ETSI) and Object Management Group (OMG) and standards that are relevant to modeling and testing CPSs under uncertainty. In this section, we first provide an overview of relevant standards and standardization bodies. Second, we summarize the procedure of selecting standardization bodies. Finally, we present a list of the selected standardization bodies in Appendix A.

2.4.1 Context

Standardization bodies are commonly classified, according to their geographical designation, into three types: international, regional and national standardization bodies [52]. International standardization bodies develop international standards. There are four most well-known and well-established international standardization bodies: the International Organization for Standardization (ISO), the International Electrotechnical Commission (IEC), the International Telecommunication Union (ITU), and the IEEE Standards Association. Under these four standardization bodies, a large number of standards have been defined. For the regional level standardization bodies, we only consider EU standards, among which the European Telecommunications Standards Institute (ETSI) produces a lot of standards in ICT. We do not include any national standardization body into the consideration, as standards produced by national level standardization bodies inherently have limited application scopes, in comparison to international and EU standards.

We aim to define methodologies to test CPSs under uncertainty. One important mean to achieve this objective is to rely on MBE technologies. Therefore, the first criterion of selecting relevant standardization bodies such as the OMG is to include standards in MBE field (C1). As we aim to devise testing methodologies, the second selection criterion is to include standardization bodies and standards that are relevant to the testing field (C2). For example, one of the highly relevant standards is the OMG's UML Testing Profile (UTP) 2 [53]. The third selection criterion is to select standardization bodies and standards that are relevant to software-intensive systems, particularly CPSs (C3).

2.4.2 Relevant standardization bodies and standards

Based on the selection criteria C1-C3 presented above, we started from screening through the standards of the standardization bodies: ISO, IEEE, IEC, JCGM, OMG, ETSI and OASIS from two perspectives: modeling uncertainty and CPS, and testing uncertainty and CPS. As the results of the first step, we pre-selected a set of standardization bodies and standards as shown in the first column of Appendix A. In the second column of Appendix A, we indicate whether a standard is for modeling, testing, Model-based Testing (MBT), or others. In the fourth column of Appendix A, we indicate whether a specific standard explicitly defines or describes Uncertainty (including Probability) and Uncertainty Measurement.

As one can see from Appendix A, in terms of modeling, OMG defines standards on system and software modeling: UML [54], SysML [55], MARTE [56], OCL [57] and MOF [58]. ISO/IEC defines UML [59], OCL [60] and KDM

[61] modeling notations, which are also defined and maintained by OMG. In addition, ISO/IEC also defines RM-ODP [62] for enabling conceptual modeling of complex systems such as CPSs.

In terms of testing and MBT, OMG defines UTP. ETSI also defines standards on model-based testing: ETSI TR 102 840 V1.2.1 [5], ETSI ES 202 951 V1.1.1 (2011-07) [63] and ETSI EG 201 015 V2.1.1 (2012-02) [64] as shown in Appendix A. ISO/IEC/IEEE 29119 [65] is a widely recognized standard for testing. In addition, ISO/IEC joined the effort to define the ISO/IEC 9646 series [66] for supporting conformance testing of OSI. As shown in Appendix A, a number of standards have been also defined in IEEE from the aspects of system and software verification and validation [67], test documentation [68], unit testing [69] and classification of Software Anomalies [70].

In Appendix A, we also include standards (from ISO, IEC and/or IEEE under Other) that are relevant to various aspects of system and software engineering, including vocabulary, architecture description, development lifecycle, risk management and assessment, and quality assurance. Particularly, we collected standards that are relevant to Uncertainty and Uncertainty Measurement, as indicated in the column "Uncertainty" of the table. ISO 61508 [71], OMG SysML [55] and MARTE [56] define concept Probability, which is one type of uncertainty measures. OMG SACM defines Evidence, Confidence and Confidence Level, which are all relevant to uncertainty and several concepts defined in U-Model. ISO/IEC and JCGM defined few standards on Uncertainty Measurement. The concept of Uncertainty and few relevant concepts are explicitly defined in ISO 31000 (as shown in Appendix A).

2.4.3 Conclusion

Based on the results of the above-presented survey, we can conclude that there does not exist any standard, which can be used as it is for modeling uncertainty and relevant concepts both subjectively and objectively. We, therefore, initiated the standardization activity of Uncertainty Modeling at OMG. Details can be found in [29].

3 UNCERTAINTY-WISE MULTI-OBJECTIVE TEST OPTIMIZATION

In this section, we detail uncertainty-wise multi-objective test optimization from the following aspects. Section 3.1 provides a formal definition for multi-objective search problem followed by formally defining uncertainty-wise multi-objective search problem (Section 3.2). Section 3.3 presents uncertainty-wise multi-objective test optimization in detail while Section 3.4 and Section 3.5 present uncertainty-wise multi-objective test set minimization and test case prioritization, respectively. Section 3.6 illustrates the above-mentioned uncertainty-wise multi-objective test set minimization and test case prioritization with examples and Section 3.7 provides a list of examples for cost-effectiveness measures based on the state-of-the-art. Last, Section 3.8 presents a set of guidelines when applying search-based techniques for addressing multi-objective test optimization problems, which is also applicable for uncertainty-wise multi-objective test optimization.

3.1 Multi-Objective Search Problem

Generically speaking, a multi-objective optimization problem has a set of objectives (*Obj*) to meet that often have trade-off relationships among them.

$Obj = \{o_1, o_2, .., o_{non}\}$, where *non* is the total number of objectives for a particular optimization problem. Each objective o_i is measured with a measure classified into one of following three categories:1) *Cost* Measures, 2) *Effectiveness* Measures, and 3) *Efficiency* Measures. Our aim is always to decrease the cost, increase the effectiveness, and/or increase the efficiency. Mathematically speaking, we have a set of *Cost* measures and *Effectiveness* Measures:

$CostMeasure = \{cm_1, cm_2, ..., cm_{ncm}\}$, where *ncm* is the total number of cost measures.

$EffectMeasure = \{em_1, em_2, ...em_{nem}\}$, where *nem* is the total number of effectiveness measures.

In addition, a set of efficiency measures can also be defined:

$EfficiencyMeasure = \{ec_1, ec_2, ..., ec_{nec}\}$, where *nec* is the total number of efficiency measures. An efficiency measure is typically calculated as *Effectiveness* per *Cost*. Assuming that all combinations of effectiveness measures (*EffectMeasure*) per cost (*CostMeasure*) are valid, then $nec = ncm*nem$. However, not all such combinations are always valid and thus $nec < ncm*nem$.

Assuming that there are a set of possible optimization solutions:

$PS = \{ps_1, ps_2, ..., ps_{nps}\}$, where *nps* is the total number of valid solutions that are typically huge and require an optimized way to the find the best solution. Let us suppose, we have three functions: 1) *CMF()* is a function that takes input a solution ps_i from *PS* and an objective measured with cost measure cm_i and returns an overall cost objective value for the solution measured with cm_i, 2) *EMF()* is a function that takes input a solution ps_i from *PS* and an objective measured with an effectiveness measure em_i and returns the overall effectiveness objective value for the solution measured using em_i, 3) *ECF()* is a function that takes input a solution ps_i from *PS* and an objective measured with an efficiency measure ec_i and returns the overall efficiency value of the solution measured using ec_i.

A multi-objective search problem then can be formally defined as finding the best solution ps_k out of *PS* such that it holds the following three conditions:

Condition 1: $\forall_{cm}$ in *CostMeasure*, $\forall_{ps}$ in $PS - ps_k$, $CMF(ps_k, cm) <= CMF(ps, cm)$. Notice that $\forall_{ps}$ in *PS* refers to the explored solutions, of all the *nps* number of solutions.

Condition 2: $\forall_{em}$ in *EffectMeasure,* $\forall_{ps}$ in $PS - ps_k$, $EMF(ps_k, em) >= EMF(ps, em)$. Once again, $\forall_{ps}$ in *PS* refers to the explored solutions, of all the *nps* number of solutions.

Condition 3: $\forall_{ec}$ in *EfficiencyMeasure,* $\forall_{ps}$ in $PS - ps_k$, $ECF(ps_k, ec) >= ECF(ps, ec)$. Once again, $\forall_{ps}$ in *PS* refers to the explored solutions, of all the *nps* number of solutions.

For instance, one of our previous works [72] employed multi-objective search (e.g., NSGA-II [73]) for investigating the resource-ware test case prioritization problem by taking into test resource (e.g., hardware) usage into account. To address such a problem, we proposed and formally defined three effectiveness objectives (i.e., test resource usage, fault detection capability and prioritization density) and one cost measure (i.e., total time). The fitness function was further defined by considering these four cost-effectiveness measures, which was further incorporated into seven multi-objective search algorithms (e.g., NSGA-II) for empirical evaluation in terms of their performance and scalability.

3.2 Uncertainty-Wise Multi-Objective Search Problem

An uncertainty-wise multi-objective search problem is a specialization of the generic multi-objective search problem, where an additional dimension of objectives is added to the search problem, i.e., *Uncertainty*. Assuming, we have a set of *Uncertainty* Measures:

UncerMeasure = $\{um_1, um_2, ..., um_{num}\}$, where *num* is the total number of measures that can be used to measure the uncertainty of a solution from different uncertainty perspectives. Assume that there is a function *UMF()* that takes input a solution ps_i from *PS* and an objective measured with an uncertainty measure um_i and returns the overall uncertainty value associated with the solution measured using um_i.

An uncertainty-wise multi-objective search problem has to meet an additional condition described below:

Condition 4: $\forall_{\text{um}}$ in *UncerMeasure,* $\forall_{\text{ps}}$ in *PS* - ps_k, *UMF*(ps_k, *um*) <= or >= *UMF*(*ps, um*). Once again, $\forall_{\text{ps}}$ in *PS* refers to the explored solutions, of all the *nps* number of solutions.

3.3 Uncertainty-Wise Multi-Objective Test Optimization

Building on the previous definitions, an uncertainty-wise multi-objective test optimization aims at optimizing a set of test cases *TC* = $\{tc_1, tc_2, ..., tc_{ntc}\}$, where *ntc* is the total number of test cases to optimize based on cost, effectiveness, efficiency, and uncertainty objectives.

3.3.1 Cost, Effectiveness, Efficiency, and Uncertainty Attributes

Test cases have a set of *Cost*, *Effectiveness*, *Efficiency*, and *Uncertainty* attributes associated with them:

CostAtt = $\{cat_1, cat_2, .., cat_{ncat}\}$, where *ncat* is the total number of cost-related attributes, and each cat_i has exactly one type that can be measured with a basic data type such as *Integer*, *Real*, and *Boolean* or an advanced data type.

EffectAtt = $\{efat_1, efat_2, .., efat_{nefat}\}$, where *nefat* is the total number of effectiveness-related attributes, and each $efat_i$ has exactly one type that can be measured with a basic data type such as *Integer*, *Real*, and *Boolean* or an advanced data type.

EfficiencyAtt = $\{ecat_1, ecat_2, .., ecat_{necat}\}$, where *necat* is the total number of efficiency related attributes and each $ecat_i$ has exactly one type that can be measured with a basic data type such as *Integer*, *Real*, and *Boolean* or an advanced data type.

UncertaintyAtt = $\{uat_1, uat_2, .., uat_{nuat}\}$, where *nuat* is the total number of uncertainty attributes and each uat_i has exactly one type that can be measured with a basic data type such as *Integer*, *Real*, and *Boolean* or an advanced data type.

3.3.2 Cost, Effectiveness, Efficiency, and Uncertainty Attribute Values

Each test case tc_i in TC has four sets of associated values that can be used to calculate the four types of objectives, i.e.,

$CostVal_i = \{cvali_1, cvali_2, ..., cvali_{ncat}\}$, where $ncat$ is the total number of cost-related attributes measured with the *Cost* measures.

$EffectVal_i = \{efvali_1, efvali_2, ..., efvali_{nefat}\}$, where $nefat$ is the total number of effectiveness related attributes measured with the *Effectiveness* measures.

$EfficiencyVal_i = \{ecvali_1, ecvali_2, \ldots, ecvali_{necat}\}$, where $necat$ is the total number of efficiency related attributes measured with the *Efficiency* measures.

$UncertaintyVal_i = \{umvali_1, umvali_2, \ldots, umvali_{nuat}\}$, where $nuat$ is the total number of uncertainty related attributes measured with the *Uncertainty* measures.

3.3.3 Calculation of cost, effectiveness, efficiency, and uncertainty objectives

In the context of a test optimization problem, the input for the search is the set of test cases, i.e., TC and its associated sets of cost, effectiveness, efficiency, and uncertainty attribute values for the ntc number of test cases. A solution ps_k in PS, in this context, would be a set of test cases from TC, i.e., either a subset (if it is a test set minimization problem) or the same subset but with the best order to execute the same set of test cases (prioritization problem).

The calculation of an objective o_i (Cost/Effectiveness/Efficiency/Uncertainty) can be performed in different ways and is often dependent on the problem being solved. A simplest and straightforward way to calculate an objective is to take an average of the solution. For example, a cost measure cm_i corresponding to a cost attribute, e.g., cat_k can be calculated as follows:

$cm_i = (\Sigma_{j=1 \text{ to } ntc}\ cvalj_k)/ntc$, where $cvalj_k$ represents a value for a j_{th} test case in TC corresponding to the cat_k cost attribute.

Similarly, for an effectiveness measure em_i corresponding to an effectiveness attribute, e.g., $efat_k$ can be calculated as follows:

$em_i = (\Sigma_{j=1 \text{ to } ntc}\ efvalj_k)/ntc$, where $efvalj_k$ represents a value for a j_{th} test case in TC corresponding to the $efat_k$ effectiveness attribute.

For an efficiency measure ec_i corresponding to an efficiency attribute, e.g., $ecat_k$ can be calculated as follows:

$ec_i = (\Sigma_{j=1 \text{ to } ntc}\ ecvalj_k)/ntc$, where $ecvalj_k$ represents a value for a j_{th} test case in *TC* corresponding to the $ecat_k$ effectiveness attribute.

For an uncertainty measure um_i corresponding to an uncertainty attribute, e.g., uat_k can be calculated as follows:

$um_i = (\Sigma_{j=1 \text{ to } ntc}\ umvalj_k)/ntc$, where $umvalj_k$ represents a value for a j_{th} test case in *TC* corresponding to the uat_k uncertainty attribute.

Given that each objective, i.e., cm_i, em_i, ec_i, and um_i, may take a value in different ranges, a common practice is to normalize the values calculated for the objectives. There are two commonly used normalization functions in the context of search-based multi-objective for software engineering problems depending on two conditions:

Condition 1: If maximum and minimum values for a calculated objective are known, then the following normalization function must be used:

$N1(x) = (x - x_{min}) / (x_{max} - x_{min})$, where x_{max} represents the known maximum value that can be taken by the variable x and x_{min} represents the known minimum value that can be taken by the variable x.

Condition 2: If the maximum and minimum values for a calculated objective are not known, then one of the following normalization functions should be used:

$N2(x) = (x/(x+\beta))$, where ß is any value greater than 0 [74, 75].

Notice that there exists another normalization function for this condition:

$N3(x) = 1- \alpha^{-x}$, where $\alpha = 1.001$ is typically used [74, 75].

However, the experiments reported in [74-76] suggest using *N2* instead of the *N1* normalization function.

Our ultimate aim is to minimize cost, maximize effectiveness and efficiency, whereas at the same time minimize (or maximize) uncertainty depending on the problem at hand, i.e.,

$\forall_{cm}$ in *CostMeasure*, Minimize(*cm*) && $\forall_{em}$ in *EffectMeasure*, Maximize(*em*) && $\forall_{ec}$ in *EfficiencyMeasure*, Maximize(ec) && $\forall_{um}$ in *UncerMeasure*, Minimize/Maximize(*um*).

3.4 Uncertainty-Wise Multi-Objective Test Set Minimization (UWTM)

In the case of *UWTM*, our aim is to find a minimum number of test cases (*mtc*) out of the total number of test cases *ntc* in *TC* that meet all the required cost, effectiveness, efficiency, and uncertainty objectives. Thus, the number of test cases (*mtc*) in a solution in *PS* can be any combination of a number of test cases in *TC*, i.e., from *1* to *ntc-1*. In this way, the possible number of minimization solutions can be calculated as follows:

$nps = {}^{ntc}C_1 + {}^{ntc}C_2 + .. + {}^{ntc}C_{ntc-1} = 2^{ntc} - 1$

$PS = \{ps_1, ps_2, ..., ps_{2\text{ntc-1}}\}$

If *mtc=ntc*, then it means that there is no minimization at all. Irrespective of which cost, effectiveness, efficiency, and uncertainty measures are selected for *UWTM*, a mandatory cost objective must be defined that considers test set minimization. Depending on the problem at hand, such an objective can be defined in different ways. One way to define such objective is to calculate the percentage of test set minimization (TMP), which can be calculated as:

TMP = 1 - ntc_i/ntc, where ntc_i is the total number of test cases in the ps_i solution in *PS* and *ntc* is the total number of test cases in *TC*.

Another way of defining a cost objective measuring test set minimization can simply be a number of minimized test cases (NMTC):

NMTC = ntc_i is the number of minimized test cases in a solution ps_i.

Given the fact that TMP is a percentage, it will produce a value between 0 and 1 and doesn't need normalization. In the case of NMTC, a normalization function may be needed to scale its value between 0 and 1. A common problem with both TMP and NMTC is that a search algorithm will favor a solution with the lower number of test cases and can potentially lead to selecting no test cases at all. In order to avoid search reaching to such a situation, various mechanisms can be employed. For example, a typical way is to control a search algorithm to produce solutions with at least *x* number of test cases, where *x* is always greater than 1.

An uncertainty-wise multi-objective test set minimization can formally be defined as finding the best solution ps_k from *PS* which has a minimum number of test cases that meet the four conditions specialized as follows:

Condition 1: $\forall_{\text{cm}}$ in *CostMeasure,* $\forall_{\text{ps}}$ in *PS* - ps_k, $CMF(ps_k, cm) <= CMF(ps, cm)$ and $mps_k < mps$ and $mps_k \neq 0$, where mps_k is the number of test cases in the ps_k solution and *mps* is the number of test cases in the *ps* solution. Notice that $\forall_{\text{ps}}$ in *PS* refer to the explored solutions of all the *nps* number of solutions.

Condition 2: $\forall_{\text{em}}$ in *EffectMeasure*, $\forall_{\text{ps}}$ in *PS* - ps_k, $EMF(ps_k, em) >= EMF(ps, em)$ and $mps_k < mps$ and $mps_k \neq 0$, where mps_k is the number of test cases in the ps_k solution and *mps* is the number of test cases in the *ps* solution. Once again, $\forall_{\text{ps}}$ in *PS* refers to the explored solutions of all the *nps* number of solutions.

Condition 3: $\forall_{\text{em}}$ in *EfficiencyMeasure*, $\forall_{\text{ps}}$ in *PS* - ps_k, $ECF(ps_k, ec) >= ECF(ps, ec)$ and $mps_k < mps$ and $mps_k \neq 0$, where mps_k is the number of test cases in the ps_k solution and *mps* is the number of test cases in the *ps* solution. Once again, $\forall_{\text{ps}}$ in *PS* refers to the explored solutions, of all the *nps* number of solutions.

Condition 4: $\forall_{\text{um}}$ in *UncerMeasure*, $\forall_{\text{ps}}$ in *PS* - ps_k, $UMF(ps_k, um)$ <= or >= $UMF(ps, um)$ and $mps_k < mps$ and $mps_k \neq 0$, where mps_k is the number of test cases in the ps_k solution and mps in the number of test cases is the *ps* solution. Once again, $\forall_{\text{ps}}$ in *PS* refers to the explored solutions, of all the *nps* number of solutions.

3.5 Uncertainty-Wise Multi-Objective Test Case Prioritization (UWTP)

UWTP is concerned with prioritizing the test cases in a specific order to execute. This is in contrast with *UWTM*, where we wanted to minimize the number of test cases without considering any order of execution. However, test set minimization and test case prioritization may be combined together. There are three possible ways: 1) Minimize test cases and then prioritize the minimized test cases, 2) Prioritize test cases and then minimize them, 3) Prioritize and minimize the test cases at the same time.

Generally speaking, when performing *UWTP*, our aim is to find a best order to execute the total number of test cases *ntc* that are available in *TC* such that it meets all the required cost, effectiveness, efficiency, and uncertainty objectives. Thus, as opposed to the test set minimization problem, where the number of test cases (*mtc*) in a solution in *PS* can be any combination of number of test cases in *TC*, *UWTP* keeps the number of test cases to the same as the total number of test cases in *TC*, i.e., *ntc*. Thus, in the case of *UWTP*, each test case can possibly appear at each location, and thus the possible number of prioritization solutions can be calculated as follows:

$nps = (ntc)*(ntc-1)*(ntc-2)*\ldots 1 = ntc!$

$PS = \{ps_1, ps_2, \ldots, ps_{ntc!}\}$

Regardless of the selection of cost, effectiveness, efficiency, and uncertainty measures for an *UWTP*, a mandatory effectiveness objective must be defined that calculates the effectiveness of prioritization. Depending on the problem at hand, such a prioritization objective can be defined in a variety of ways. One example to calculate the effectiveness of prioritization is using Prioritization Impact (*PI*) of a location of an order of a test case, which can be calculated as follows:

$PI_i = (ntc-p+1)/ntc$, where p represents the p_{th} location in a prioritization solution. In other words, the test cases with low cost, high effectiveness, high efficiency, and low uncertainty must be ordered to execute at the earlier positions. For example, at the first location, i.e., $p=1$, $PI_i=1$, whereas at $p=ntc$, $PI_i= 1/ntc$. Such number is already scaled between 0 and 1, and thus doesn't require normalization. A higher value means a high priority.

Another way of defining such an effectiveness objective measuring prioritization impact can simply be the position, i.e., p, where p ranges from 1 to *ntc* and we aim to order the best test case in terms of cost, effectiveness, efficiency, and uncertainty in the earlier position. The fact that p is a number greater than 1, indicates that normalization of p between 0 and 1 may be required. In this particular case, since we know the maximum (i.e., *ntc*) and minimum (i.e., 1) values for the position, we can use the following normalization function:

$N1(p) = (p) / (ntc - 1)$

Typically, when performing test case prioritization, it is not possible to execute all the test cases (i.e., *ntc*). The execution of the number of test cases is limited by time budget, for example, a certain percentage of test cases. Such time budget can be fixed and encoded in the search problem as an additional constraint that must be satisfied by a solution. Such time budget can be calculated in different ways. For example, as a certain percentage of test cases, e.g., 10% or 20%. Another example is to set this time budget based on the actual execution time, such as in terms of the maximum number of hours.

An uncertainty-wise multi-objective test set prioritization problem can formally be defined as finding the best solution ps_k from *PS,* which has test cases in the best order to execute such that it meets the following four conditions specialized for *UWTP*:

Condition 1: $\forall_{\mathrm{cm}}$ in *CostMeasure*, $\forall_{\mathrm{ps}}$ in *PS* - ps_k, *CMF*(ps_k, *cm*) <= *CMF*(*ps*, *cm*) and *tbg* <= *bg*, where *tbg* represents the overall time budget for the ps_k solution and *bg* represents the available time budget for test case prioritization. Notice that $\forall_{\mathrm{ps}}$ in *PS* refer to the explored solutions of all the *nps* number of solutions.
Condition 2: $\forall_{\mathrm{em}}$ in *EffectMeasure*, $\forall_{\mathrm{ps}}$ in *PS* - ps_k, *EMF*(ps_k, *em*) >= *EMF*(*ps, em*) and *tbg* <= *bg*, where *tbg* represents the overall time budget for the ps_k solution and *bg* represents the available time budget for test case prioritization. Once again, $\forall_{\mathrm{ps}}$ in *PS* refers to the explored solutions, of all the *nps* number of solutions.
Condition 3: $\forall_{\mathrm{em}}$ in *EfficiencyMeasure*, $\forall_{\mathrm{ps}}$ in *PS* - ps_k, *ECF*(ps_k, *ec*) >= *ECF*(*ps, ec*) and *tbg* <= *bg*, where *tbg* represents the overall time budget for the ps_k solution and bg represents the available time budget for the test case prioritization problem. Once again, $\forall_{\mathrm{ps}}$ in *PS* refers to the explored solutions of all the *nps* number of solutions.
Condition 4: $\forall_{\mathrm{um}}$ in *UncerMeasure*, $\forall_{\mathrm{ps}}$ in *PS* - ps_k, *UMF*(ps_k, *um*) <= or >= *UMF*(*ps*, *um*) and *tbg* <= *bg*, where *tbg* represents the overall time budget for the ps_k solution and *bg* represents the available time budget for the test case prioritization problem. Once again, $\forall_{\mathrm{ps}}$ in *PS* refers to the explored solutions, of all the *nps* number of solutions.

3.6 Examples of UWTM and UWTP

In this section, using the formalism defined from Section 3.1 to Section 3.5, we will provide *UWTM* and *UWTP* examples. The both examples will demonstrate the applications of *UWTM* and *UWTP* for testing an industrial CPS case study. The need for these *UWTM* and *UWTP* arose in the context of a project, where we are developing new methods to test CPSs in the presence of uncertainty [77]. Our overall solution was Uncertainty-Wise Model-Based Testing (*UWMBT*) as defined in Section 2.2. The expected behavior of a CPS under test together with uncertainty was modeled with *UncerTum* (Section 2.2.1) and then using our tool support a set of abstract test cases was generated [33]. Given the complexity of models, the number of generated test cases was large and it was practically impossible to execute all the generated test cases, and thus required the development of not only *UWTM* but also *UWTP*. In the following sub-sections, we will present the *UWTM* and *UWTP* solutions that we developed as examples. Their details and evaluations are reported in [33, 78] and interested readers are suggested to consult these references for further information. The rest of this section is organized as follows: In Section 3.6.1, we present the example of *UWTM* and example of *UWTP* in Section 3.6.2.

3.6.1 Example of UWTM

In our previous work reported in [33], we defined four *UWTM* problems and demonstrated their application for an industrial CPS case study. Test cases, in this case, were generated from stereotyped UML State machines as discussed in Section 2.2.1 using different uncertainty-wise test generation strategies including All Simple Paths with Uncertainty (ASP) and All Paths with Uncertainty and a Fixed Maximum Length (APML) as discussed in Section 2.2.2 and in [33]. Each generated test case had sets of associated *EffectiveAtt* and *UncertaintyAtt* attributes as listed below:

EffectiveAtt = {ntran}, where *ntran* represents the number of transitions covered by a test case

UncertaintyAtt = {nun, nuun, um), nun represents the number of uncertainties in a test case, *nuun* represents the number of unique uncertainties in a test cases, and *um* represents the overall uncertainty of a test case calculated using the uncertainty measure defined in Uncertainty Theory [41].

Notice that it is not always necessary to have all four kinds of attributes (*Cost*, *Effectiveness*, *Efficiency*, and *Uncertainty* as discussed in Section 3.3) associated with test cases. In this particular *UWTM*, we defined the following *Cost*, *Effectiveness*, and *Uncertainty* Objectives:

Obj = {PTM, ANU, PUS, AUM, PUU, PTR}

CostMeasure ={PTM}, where *PTM* is the same as *TMP* as defined in Section 3.4 and measures the percentage of minimization.

EffectMeasure = {PTR}, where *PTR* is an effectiveness measure used to calculate the overall transition coverage achieved by a minimized solution. The formula for calculating *PTR* can be found in [33].

UncerMeasure = {ANU, PUS, AUM, PUU}, where *ANU* is an uncertainty measure used to calculate an average number of uncertainties covered by a minimized solution. *PUS* is another uncertainty measure used to calculate the overall uncertainty space covered by a minimized solution. Notice that the concept of uncertainty space is defined in uncertainty theory [41] and is adopted in our work. The formula for calculating *PUS* can be found in [33]. *AUM* is used to calculate the overall average uncertainty measure of a minimized test set. Recall that uncertainty measure is defined in Uncertainty theory [41] and is adopted in our work. The formula for calculating *AUM* can be found in [33]. The fourth uncertainty measure we defined is *PUU,* which is used to calculate the overall average number of unique uncertainties covered by a minimized solution. The formula for *PUU* can be consulted in [33].

Given that not all the uncertainty measures can be used at the same time, we developed four UWTM problems described below:

$UWTM_1$ = CostMeasure {PTM}, EffectMeasure = {PTR}, and UncerMeasure = {ANU}

$UWTM_2$ = CostMeasure {PTM}, EffectMeasure = {PTR}, and UncerMeasure = {PUS}

$UWTM_3$ = CostMeasure {PTM}, EffectMeasure = {PTR}, and UncerMeasure = {AUM}

$UWTM_4$ = CostMeasure {PTM}, EffectMeasure = {PTR}, and UncerMeasure = {PUU}

In the above four *UWTM*s, the cost objectives must be minimized, whereas the effectiveness and uncertainty objectives must be maximized. Depending on the definitions of uncertainty measures, uncertainty objectives may be minimized or maximized. In our context, all the uncertainty objectives must be maximized, e.g., we aim to cover as much uncertainty as possible with a minimized solution.

A variety of multi-objective search algorithms may be used to solve our *UWTM*s. As an initial evaluation, we opted for the most commonly used multi-objective search algorithm, i.e., NSGA-II [73] to solve our *UWTM*s. We used its implementation in the jMetal framework [80]. We empirically evaluated the four UWTMs using an open source case study of SafeHome, where we compared the four *UMTM*s in terms of their effectiveness with mutation testing. Based on the results of our experiment, we found that *$UWTM_4$* was able to find the best-minimized solution in terms of minimized number of test cases and mutation score (i.e., the number of seeded faults found). Notice that these minimized number of test cases and mutation score are comparable across all the four *UWTM*s. *$UWTM_4$* managed to achieve *PTM* of 91%, whereas it managed to achieve 100% mutation score [33].

With the best *UWTM*, i.e., *$UWTM_4$*, we minimized the number of test cases for the industrial CPS case study of GeoSports (GS) [81]. We used one use case focusing on testing the implementation of GS in the presence of uncertainties and generated test cases using our *UWMBT* technique with the tool presented in [33]. With *$UWTM_4$*, we managed to minimize 83.9% of test cases and managed to discover 98 uncertainties when executing the minimized test cases. 18 out of these 98 were newly discovered uncertainties. All the details of *UWTM*s and empirical evaluations can be consulted in [33].

3.6.2 Example of UWTP

Our previous work [78] reported an Uncertainty-wise Test Case Prioritization Problem (*UWTP*) that we defined and demonstrated its application in an industrial CPS case study. The same as the *UWTM*s, test cases were obtained using our *UWMBT* techniques as described in Section 2.2.2 and full details in [78]. Each generated test case had the following sets of associated *CostAtt*, *EffectiveAtt*, and *UncertaintyAtt* attributes listed below:

CostAtt = {etime}, where *etime* is the execution time for a test case

EffectiveAtt = {ntran}, where *ntran* represents the number of transitions covered by a test case

UncertaintyAtt = {oun, um}, where *oun* represents the number of uncertainties observed as the result of execution of the test case and *um* represents the overall uncertainty of a test case calculated using the uncertainty measure defined in Uncertainty Theory [41].

The same as any other multi-objective test optimization problem, it is not mandatory to have all the four types of attributes that we defined. In our *UWTP*, we defined the following *Cost*, *Effectiveness*, and *Uncertainty* objectives:

Obj = {PET, ANOU, AUM, PTR}

CostMeasure = {PET}, where *PET* is the overall execution time for the prioritized set of test cases. The formula for calculating *PET* can be found in [78]. Notice that in this work, we had access to the execution time of each test case; therefore, we opted for time budget based on the overall execution of test cases. For example, we wanted to prioritize test cases based on *x*% of the total time budget, where total time budget was equal to the total time to execute all the *ntc* test cases.

EffectMeasure = {PTR}, where *PTR* is an effectiveness measure used to calculate the overall transition coverage achieved by a minimized solution. The formula for calculating *PTR* can be found in [78].

UncerMeasure = {ANOU, AUM}, where *ANOU* is an uncertainty measure used to calculate an average number of observed uncertainties in the last execution of the test cases contained in a prioritized solution. The formula for calculating *ANOU* can be found in [78]. *AUM* is used to calculate the overall average uncertainty measure of a prioritized test set. Recall that uncertainty measure is defined in Uncertainty theory and is adopted in our work. The formula for calculating *AUM* can be found in [78].

In terms of *PI*, i.e., Prioritization Index, we used the formula presented in Section 3.5, i.e., *(ntc-p+1) /ntc*. Our uncertainty-wise multi-objective prioritization problem was represented as follows:

UWTP = CostMeasure = {PET}, EffectMeasure = {PTR}, and UncerMeasure = {ANOU, AUM}

In the above *UWTP*, the cost objective must be minimized, and the effectiveness objective must be maximized, whereas uncertainty objectives must be maximized as well, i.e., we wanted to prioritize the solutions based on both the modeled subjective uncertainties and the observed uncertainties that were discovered as the result of the last test execution.

In terms of evaluation, we used the same industrial CPS case study, i.e., GeoSports as our *UWTM*s. Once again we selected NSGA-II for solving our *UWTP*; although we also compared the performance of NSGA-II with a greedy algorithm as the comparison baseline [78]. In addition, to assess the scalability of NSGA-II in terms of solving *UWTP* problems with varying complexities, we also simulated, in total, 72 problems ranging from simpler to the complex problems. The results of our evaluation showed that NSGA-II significantly outperformed Greedy for the industrial case study and also for the 72 simulated problems. We concluded that, on average for both industrial and simulated problems, NSGA-II improved prioritization by 22% as compared to the Greedy algorithm. These results showed that NSGA-II is cost-effective and scalable for solving our Uncertainty-Wise Test Case Prioritization problem for a variety of problems [78].

3.7 Examples of Cost and Effectiveness Measures

In this section, we present some examples of cost, effectiveness, and uncertainty objectives from the literature, which are related to multi-objective test optimization. Table 2 presents the name and a short description of these objectives together with references where the details can be found.

Table 2. List of cost, effectiveness, and uncertainty objectives

Label	Name	Description
Effectiveness		
E1	The percentage of reduction in test set size (*TMP*)	*TMP* measures the percentage of reduction in test set size compared with original test set [83]. For example, see its use in [83, 84].
E2	The percentage of reduction in fault detection by selected test cases	The number of faults detected by the original test set *TC* is *nf*, whereas the number of faults detected by the optimized test cases is *mf*. Such reduction in fault detection [83] is calculated as *1–(mf/nf)*100*%.
E3	The rate of fault detection	"A measure of how quickly a test set detects faults during the testing process" [85]. Such measure is used with test case prioritization.
E4	Coverage of statements of code (Total coverage and additional coverage)	"A measure of the percentage of statements that have been executed by test cases" [86, 87].
E5	Coverage of event (Total coverage and additional coverage)	"A measure of the percentage of events that have been executed by test cases" [88, 89].
E6	Coverage of call-stack	"A test set is represented by a set of unique maximum depth call stacks; its minimized test set is a subset of the original test set whose execution generates the same set of unique maximum depth call stacks" [90].
E7	Operational coverage	"An operational abstraction is a formal mathematical description of program behavior: it is a collection of logical statements that abstract the program's runtime operation. Operational coverage (defined below) measures the difference between an operational abstraction and an oracle or goal specification." [86]
E8	*def-use* Coverage	"A measure of the percentage of definitions that have been executed by test cases." [79, 91]
E9	Coverage of Branch (Total probability and additional probability)	"A measure of the percentage of branches that have been executed by test cases." [86, 91, 92]
E10	Coverage of Block (Total probability and additional probability)	"A measure of the percentage of blocks that have been executed by test cases." [93]
E11	Coverage of Functions (Total coverage and additional coverage)	"A measure of the percentage of functions that have been executed by test cases." [85]
E12	Coverage of Interaction (Total probability and additional probability)	"A measure of the percentage of interactions that have been executed by test cases." [88, 89]
E13	Percentage of Transition Coverage (*PTR*)	*PRT* measures the percentage of the total number of unique transitions covered by the minimized [33] or prioritized [78] subset of test cases.
E14	Probability of exposing faults (Total probability and additional probability)	*FEP* [85, 94] is the commonly used metric to measure the probability of exposing faults is fault-exposing-potential. It can be calculated as: *FEP= km/tm*, where *km* is the number of killed mutants by *TC*, and *tm* is total mutants in *TC*.
E15	Probability of faults existence (Total probability and additional probability)	"Faults are not equally expected to exist in each function; rather, certain functions are more liable to contain faults than others. This fault proneness can be associated with measurable software attributes. Test cases based on their history of executing fault-

Label	Name	Description
		prone functions are prioritized taking the advantage of their association." [95] A metric to measure as a metric of fault proneness is a fault index [95].
E16	Average Percentage of Faults Detected (*APFD*) (only prioritization)	*APFD* [85, 96] measures the average percentage of faults detected after prioritization. It can be calculated as: $APFD = 1-(\sum_{i=1\ to\ nf} ps(i))/(nf*ntc)+1/(2ntc)$, where *ntc* is number of test cases in *TC*, $F = \{f_1, \ldots, f_{nf}\}$ is the set of *nf* faults revealed by *TC*, and $ps(i)$ is the first position of test cases in the prioritized test cases which reveals fault f_i .
E17	Unit-of-fault-severity-detected-per-unit-test-cost *APFDc* (only prioritization)	$APFD_c$ [97] measures the average of percentage of faults detected with cost after prioritization.
E18	Customized Test Requirement	There are some characteristics of requirement of test cases, e.g., importance [98], risk[99], customer-assigned priority [98, 100, 101], implementation complexity (CI) [98, 101], requirement changes [101], completeness (CT) [101], traceability (TR) [101], dependency [102], scope [103], property relevance [104].
E19	Feature Pairwise Coverage (*FPC*)	*FPC* is defined to measure how many feature (testing functionalities) pairs can be achieved by a produced solution [84].
E20	Fault Detection Capability (*FDC*)	*FDC* measures the fault detection capability of an obtained solution. In [84], fault detection is defined as the successful execution rate (manage to detect faults) for a test case in a given time, e.g., a week.
E21	Average Execution Frequency (*AEF*)	*AEF* measures the average execution frequency of a solution during a given time (e.g., a week) using the execution frequency of each included test cases [84].
E22	Prioritization Density (*PD*)	*PD* measures how many test cases can be prioritized by a given solution with available test resources (e.g., hardware) [72].
E23	Test Resource Usage (*TRU*)	*TRU* is defined to measure how many available test resources (e.g., hardware) can be used by a solution [72].
E24	Mean Priority (*MPR*)	*MPR* measures the importance of the test cases in a solution, which is determined based on the type of test requirements [105].
E25	Mean Probability (*MPO*)	*MPO* measures the likelihood that a solution (including a number of test cases) in terms of detecting faults [105].
E26	Mean Consequence (*MC*)	*MC* measures the impact of a failure of the test cases in a solution that the system can have on the environment [105].
E27	Configuration Coverage (*CC*)	*CC* measures the overall configuration coverage of a solution with a set of a number of test cases [106].
E28	Test API Coverage (*APIC*)	*APIC* measures the overall test API coverage of a solution [106].
E29	Status Coverage (*SC*)	*SC* measures the overall status coverage of a solution [106].
Cost		
C1	Saving Factor, SF (unit is dollars)	This is measured based on "savings in time are associated with savings in dollars through engineer salaries, accelerated business opportunities" [72].
C2	Overall Execution Time (*OET*)	*OET* measures the total execution time for a solution (including a number of test cases) based on the historical execution data [84].
C3	Number of test cases (*NMTC*)	*NMTC* measures the number of test cases that need to be executed. For example, see its use in [33].
C4	Number of statements	A measure of the number of statements that have been executed by test cases. For example, see its use in [96].
C5	Total Time (*TT*)	*TT* measures the total time cost for a solution including test case execution time and test resource allocation time [72].
C6	Time Difference (*TD*)	*TD* measures the difference in time between a pre-given time budget and execution time for a solution (that consists of a number of test cases) [105].
Additional Measures		
A1	Average Normalized Number of Uncertainties Covered (*ANU*)	*ANU* is used to measure the average normalized number of uncertainties covered by the minimized number of test cases [33].
A2	Percentage of Uncertainty Space Covered (*PUS*)	*PUS* measures the percentage of the total set of uncertainty spaces of a belief state machine [21] covered by the minimized subset of test cases [33].

Label	Name	Description
A3	Average Overall Uncertainty Measure (*AUM*)	*AUM* measures the average overall uncertainty of the minimized [33] or prioritized [78] subset of test cases.
A4	Percentage of Unique Uncertainties Covered (*PUU*)	*PUU* measures the percentage of the total number of unique uncertainties covered by the minimized subset of test cases [33].
A5	Average Number of Observed Uncertainties (*ANOU*)	*ANOU* measures the average number of observed uncertainties by the prioritized test cases [78].

3.8 Analyzing Results

In this section, we discuss a set of guidelines extracted based on our previous experience of applying SBST to address various multi-objective test optimization problems including uncertainty-wise multi-objective test optimization. We present these guidelines from the following perspectives, which include: 1) *fitness function* that discusses what should be paid attention when defining and formulating fitness function; 2) *multi-objective algorithms* that discuss how to select and compare multi-objective search algorithms; 3) *evaluation metrics* that present how to select appropriate evaluation metrics for evaluating the performance of multi-objective search algorithms; and 4) *statistical tests* that discuss how to select proper statistical tests for analyzing the results obtained by the multi-objective search algorithms in conjunction with the defined fitness function. It is worth mentioning that the summarized guidelines are considered as generic recommendations, which are applicable to any multi-objective test optimization problems (including uncertainty-wise multi-objective test optimization problems).

3.8.1 Fitness function

It is well recognized that a fitness function is defined for assessing the quality of solutions obtained during the search, and thus defining an appropriate fitness function is of paramount importance to obtain optimal solutions when addressing multi-objective test optimization problems. The first key step to define a proper fitness function is to formally formulate the required objectives as a set of cost-effectiveness measures based on the specific domain knowledge. For instance, to tackle the multi-objective test set minimization problem [84], we discussed with the test engineers of our industrial partner (with the telecommunication domain) and proposed five objectives that should be taken into account. These five objectives were further formulated as four effectiveness measures (i.e., objective functions) and one cost measure, which includes 1) *test set minimization* to measure the amount of reduction in terms of the number of test sets, 2) *feature pairwise coverage* to measure how many feature (testing functionalities) pairs can be achieved by a solution, 3) *fault detection capability* to measure the capability of detection faults based on the historical data; 4) *average execution frequency* that measures how often the minimized test cases can be executed based on the execution history; and 5) *overall execution time* to measure the overall time taken for executing the minimized test cases.

It is worth mentioning that it is practically possible to further refine the defined cost-effectiveness measures based on particular restrictions from particular domains, and thus it is recommended to first finalize the cost-effectiveness measures (objective functions) before defining fitness function (**Recommendation, R1**). For instance, with respect to the effectiveness measure fault detection capability, it is common in the existing literature to define such measure using the number of detected faults (bugs). However, in the context of our case (i.e., telecommunication domain) [84], such information related with the number of faults is not available from the historical execution data, and thus we proposed to measure the fault detection capability by calculating the number of successful executions out of the total number of executions (a successful execution means faults can be detected by executing a specific test case). The measurement of fault detection capability was further discussed, agreed and finalized with the test engineers from our industrial partner. Therefore, we recommend practitioners defining the cost-effectiveness measures based on the particular domains since it is sometimes possible that the common measures from the state-of-the-art cannot fulfill the practical requirements from the domains (**R2**).

Once the cost-effectiveness measures are formally formulated, the next step is to incorporate these measures into a fitness function that is used to guide the search towards finding optimal solutions. Based on the state-of-the-art, the fitness function can be defined using two means including 1) directly using the defined cost-effectiveness measures as the fitness function when the Pareto-based multi-objective algorithms (e.g., NSGA-II) are further employed in conjunction with the fitness function; and 2) assigning particular weights to each cost-effectiveness measure and converting the multi-objective test optimization problem to a single-objective test optimization problem if the weight-based multi-objective algorithms are applied (e.g., RWGA). One general guide to recommend is that the first mean should be applied if all the objectives hold equivalent priority, i.e., the objectives are considered as equally important and the second mean is chosen if there are specific priorities (i.e., user preferences) among the objectives in practice, e.g., fault detection capability is more important than the test minimization percentage for the test set minimization problem (**R3**). Notice that the performance of Pareto-based search algorithms will be significantly decreased when the number of cost-effectiveness measures (objectives) is more than six [107] and thus it is suggested to limit the number of objectives less or equal to six if the Pareto-based search algorithms are to be applied (**R4**). However, it is worth mentioning that one of our previous works [108] proposed a new Pareto-based search algorithm (named UPMOA) to incorporate user preferences into NSGA-II, which made it possible for the Pareto-based search algorithms to handle the multi-objective test optimization problems with various user preferences.

In addition, if the weight-based search algorithms are to be employed, there are three weight assignment strategies defined and applied in the existing literature [109], which include: 1) *Fix Weights* (*FW*) assignment strategy that assigns pre-defined quantitative weights (between 0 to 1) to each objective (e.g., 0.5 to the fault detection capability) based on the user preferences from domain experts (e.g., test engineers); 2) *Randomly-Assigned Weights* (*RAW*) assignment strategy that randomly generates normalized weights (between 0 and 1) for each test optimization objective that satisfy the defined user preferences; and 3) *Uniformly Distributed Weights* (*UDW*) assignment strategy that generates normalized weights for objectives, which 1) meet the user preferences; and 2) guarantee the uniformity for the generated weights with the aim of ensuring that each search direction is explored with equal chances. The experiment results from [109] showed *UDW* can manage to achieve better performance as compared with the other two weight assignment strategies (i.e., *FW* and *RAW*), which can be used as a guide when there is a need to apply these weight-assignment strategies for weight-based genetic algorithms (**R5**).

3.8.2 Multi-objective search algorithms

After the fitness function is properly defined, the next step is to select multi-objective search algorithm to incorporate the fitness function before running the search. As mentioned before, Pareto-based and weight-based multi-search algorithms can be used and all these algorithms have intrinsic randomness because of several factors such as initial random population, the use of a crossover, and mutation operators. Thus, it is recommended to repeat each algorithm at least 10 times [74, 107], which can ensure that the results obtained by the search algorithms are not achieved at random and thereby reducing the random variations inherited in the search algorithms (**R6**). Such recommendation can be used as a general guide to any multi-objective test optimization problems at hand including uncertainty-wise test optimization.

The selection of a multi-objective search algorithm for dealing with a specific problem is also an important perspective that may have a huge impact on the performance of a search-based approach. Most of the existing works chose NSGA-II for solving multi-objective test optimization problems (e.g., [106] for test case prioritization, [110] for test case selection) and NSGA-II has proven to achieve promising results for addressing various multi-objective test optimization problems. However, several existing works have shown that Random-Weighted Genetic Algorithm (RWGA) can achieve better performance than NSGA-II, e.g., test set minimization problem [84]. Therefore, how to select the most appropriate multi-objective search algorithms largely depends on particular optimization problems and we would recommend applying and comparing at least NSGA-II (Pareto-based) and RWGA (weight-based) when solving a multi-objective test optimization problem when all the objectives are equally important (**R7**). However, as discussed before, if the objectives hold certain priories (i.e., user preferences), UPMOA [108] or RWGA are recommended to apply and compare since the traditional Pareto-based search algorithms (e.g., NSGA-II) cannot handle the multi-objective test optimization problems with user preferences (**R8**).

Furthermore, to assess the performance and justify the applicability of a multi-objective search algorithm, it is usually recommended to compare the chosen algorithms with at least, one baseline algorithm for a sanity check (e.g., random search or greedy algorithm) (**R9**) [74, 84, 107, 108, 111, 112]. Such practice can prove that the problem to be addressed at hand is complex enough, which requires the applications of an advanced multi-objective algorithm (e.g., NSGA-II). This recommendation is once again a generic one and applicable to any multi-objective test optimization problems, and even single objective test optimization problems.

Moreover, another important decision to make while employing multi-objective search algorithms lies in setting parameters for the selected algorithms. A typical practice is to use the default parameter settings that are for example provided by the jMetal library (e.g., crossover rate is 0.9, mutation probability is set as *1/n* where *n* is the total number of variables (e.g., test cases) and population size is 100) [105, 106, 112]. Such default settings have often provided good results and thus it is recommended to use at least the default settings for the selected multi-objective search algorithms (**R10**). Notice that parameter tuning is an alternate technique to find the best parameter settings for the multi-objective search algorithms but parameter tuning is quite expensive in practice [75], which is considered as an optional recommendation if there are sufficient resources (e.g., manual effort) in practice for such parameter tuning activity. Interested readers can consult [75] for more details in terms of some parameter tuning techniques and key observations for search-based software engineering.

3.8.3 Evaluation metrics

It is critical to employ proper evaluation metrics for evaluating the performance of multi-objective search algorithms along with the defined fitness function. The evaluation metrics can be chosen when different types of multi-objective search algorithms are applied, i.e., weight-based search algorithms and Pareto-based search algorithms. In terms of weight-based multi-objective search algorithms, it is recommended to directly use the obtained fitness function values as evaluation metrics for assessing the performance of algorithms (**R11**). Recall that weight-based search algorithms convert a multi-objective test optimization problem into a single-objective test optimization problem by assigning weights to each objective. When the search process is terminated, one fitness function value is produced that represents the quality of the corresponding solution. In addition, if users are also interested in particular individual objective (e.g., fault detection capability), it is also possible to compare the solutions obtained by search algorithms using each individual objective function value. However, notice that only the fitness function value can represent the overall quality of a solution obtained by a search algorithm.

With respect to the Pareto-based multi-objective search algorithms, it is recommended to employ standard quality indicators (e.g., hypervolume (*HV*), Epsilon (ϵ), Generalized Spread (*GS*), Generational Distance (*GD*)) as evaluation metrics to evaluate the performance of algorithms (**R12**). Our previous work [107] has proposed a practical guide in terms of selecting quality indicators for assessing the performance of Pareto-based multi-objective search algorithms based on theoretical foundations, literature review and extensive empirical experiment, and thus it is always recommended to use this guide for choosing quality indicators when there is a need for Pareto-based search algorithm evaluation (**R13**). We briefly summarize the guide as below to make the book chapter self-contained. More details related to such guide can be consulted in [107].

1. When a user is only concerned whether the obtained solutions are optimal or not (*Convergence*): If an ideal set of objective values (i.e., optimal objective values such as the best value for the objective of test minimization percentage is 0 when all the test cases are eliminated) is unknown for a multi-objective optimization problem, any of the quality indicators from *GD*, Euclidean Distance from the Ideal Solution (*ED*), ϵ and Coverage (*C*) is recommended. On the opposite, *ED* is suggested if an ideal objective set is known before.

2. When a user is also concerned with the *Diversity* of the obtained solutions in addition to the convergence: When the number of objectives for a particular multi-objective optimization problem is more than six, Inverted Generational Distance (*IGD*) is recommended. Otherwise, *HV* is suggested if the number of objectives is less than or equal to six.
3. When a user prefers evaluating the performance of Pareto-based search algorithms from the two perspectives of *Convergence* and *Diversity* separately: Such situation may sometimes occur since separate quality indicators to measure convergence and diversity may be more accurate than the combined quality indicators, e.g., *HV* and *IGD* (mentioned as above). In this case, if the ideal objective set is known, *ED* together with Pareto front size (*PFS*) and *GS* are recommended. Otherwise, any of the quality indicators from *GD*, *ED*, ϵ and *C* together with *PFS* and *GS* can be applied. It is worth mentioning that assessing the performance of Pareto-based multi-objective search algorithms from the two separate perspectives makes it difficult to draw a definite and clear conclusion which multi-objective search algorithm is better and thus we usually recommend selecting the combined quality indicators from (i.e., *HV* or *IGD*) with the aim of assessing the algorithm performance.

In addition, the running time taken by the search algorithms (both weight-based and Pareto-based multi-objective search algorithms) is also recommended to consider when evaluating the performance of algorithms since it would be practically infeasible to apply a search-based approach if it took the unacceptable running time to obtain optimal solutions (**R14**).

3.8.4 Statistical tests

Recall that search algorithms are usually required to be run at least 10 times due to the random variations inherited and a large amount of data results will be obtained when the search process is finished. Thus, it is a common practice to employ standard statistical tests to analyze the results obtained and extracts the key observations. Notice that the following described statistical tests are generic and applicable to all the multi-objective search algorithms and any multi-objective optimization problems.

First, it is recommended to perform the descriptive statics (e.g., mean, median and standard derivation) with the aim of acquiring direct impressions how good the results are (with the values of mean and median) and how the results are distributed (with the values of standard derivation) (**R15**). Moreover, we recommend performing the Vargha and Delaney statistics (A_{12}), which is a non-parametric effect size measure (**R16**). More specifically, in the context of SBSE, A_{12} is usually employed to measure the probability of yielding higher values for each objective function and fitness function value when comparing two multi-objective algorithms *A* and *B*. The two algorithms are considered to have equivalent performance if A_{12} is 0.5. The algorithm *A* has higher chances to obtain better solutions than the algorithm *B* if the value of A_{12} is greater than 0.5.

In addition, to compare the difference significance of several multi-objective search algorithms, it is recommended to perform a set of significance tests discussed as follows (**R17**). First, we recommend performing the Kruskal–Wallis test together with the Bonferroni Correction [113] with the aim of determining if there are significant differences among multiple sets of results (samples) obtained by multi-objective search algorithms. Notice that the Bonferroni Correction is used to adjust the *p*-value obtained by the Kruskal–Wallis test [113, 114]. If the adjusted *p*-value of the Kruskal–Wallis test is more than 0.05, it indicates that there are no significant differences observed among the results obtained by various search algorithms and thus it can be concluded that the performance of search algorithms is statistically similar.

If the adjusted *p*-value is less than 0.05 that indicates that there exist significant differences among the results among by different search algorithms, the following statistical tests are recommended. First, the Shapiro-Wilk test [114] is recommended to determine the normality of obtained data samples with the aim of selecting an appropriate statistical test for checking the significance of the difference. The significance level is usually set as 0.05, i.e., a data sample is normally distributed if the *p*-value from the Shapiro-Wilk test is greater than 0.05. On the condition that the sample data is normally distributed, it is recommended to perform *t*-test (parametric test) [113] for determining whether there exist significant differences between the results obtained by different search algorithms. When the sample data is far away from a normal distribution, the Mann-Whitney U test (non-parametric test) [113] is suggested to determine the significance of the difference between the results obtained by different search algorithms. Similarly, the significance level is usually set as 0.05, i.e., there is a significant difference observed between the obtained results if the *p*-value is less than 0.05.

In addition, to assess the scalability of the multi-objective search algorithms, we recommend the Spearman's rank correlation coefficient (ρ) [113] to measure the correlations between the results obtained by the algorithms and the complexity of problems (**R18**). The value of ρ ranges from -1 to 1, i.e., a value greater than 0 denotes a positive correlation between the results of search algorithms and the complexity of problems while a negative correlation is observed when the value of ρ is less than 0. A value of ρ close to 0 indicates that there is no correlation between the results obtained by search algorithms and the complexity of problems. Moreover, it is also recommended to report significance of correlation using $Prob>|\rho|$, i.e., a value that is lower than 0.05 means that the observed correlation is statistically significant. For instance, with respect to addressing the test set minimization problem, our previous work [84] assessed the scalability of ten multi-objective search algorithms by measuring the correlation (using the Spearman's rank correlation coefficient) between the obtained results and the increasing complexity of problems measured by the number of features (i.e., testing functionalities of systems) included. The results showed that the performance of most of the multi-objective search algorithms was not significantly decreased when the problems become more complex when dealing with our test set minimization problem.

In summary, this section presents in total 18 recommendations (**Rs**) as shown in Table 3 that can be used by researchers and practitioners when there is a need to apply multi-objective search algorithms to tackle uncertainty-wise multi-objective test optimization problems.

Table 3. Summary of the Recommendations

R #	Category	Description
1	Fitness Function	It is recommended to first finalize cost-effectiveness measures (objective functions) before defining fitness function.
2		It is recommended to define cost-effectiveness measures based on particular domains to fulfill practical requirements.
3		It is recommended to directly use objective functions as fitness function when Pareto-based multi-objective algorithms are applied. If weight-based multi-objective algorithms are applied, it is recommended to assign weights to each objective function and convert the multi-objective test optimization problem to single-objective test optimization problem.
4		It is recommended to limit the number of objectives less or equal to six if Pareto-based search algorithms are to be applied.
5		It is recommended to apply Uniformly Distributed Weights (*UDW*) assignment strategy when there is a need to use weight-based genetic algorithms.
6	Multi-Objective Search Algorithms	It is recommended to repeat each multi-objective search algorithm at least 10 times for reducing the random variations inherited in search algorithms.
7		It is recommended to apply and compare at least NSGA-II (Pareto-based) and RWGA (weight-based) when solving a multi-objective test optimization problem when all the objectives are equally important.
8		It is recommended to apply and compare UPMOA or RWGA if the objectives hold certain priorities (i.e., user preferences).
9		It is recommended to compare the selected multi-objective search algorithms with at least one baseline algorithm for a sanity check (e.g., random search or greedy algorithm).
10		It is recommended to use at least the default settings for the selected multi-objective search algorithms.
11	Evaluation Metrics	It is recommended to directly use the obtained fitness function values as evaluation metrics for assessing the performance of weight-based multi-objective search algorithms.
12		It is recommended to employ standard quality indicators (e.g., hypervolume (HV), Epsilon (ϵ)) as evaluation metrics to evaluate the performance of Pareto-based multi-objective search algorithms.
13		It is recommended to use the guide from [107] for choosing quality indicators when there is a need for evaluating the performance of Pareto-based search algorithms.
14		It is recommended to compare the running time taken by the search algorithms when evaluating the performance of algorithms.
15	Statistical Tests	It is recommended to perform the descriptive statics (e.g., mean, median and standard derivation) for acquiring direct impressions of the results.
16		It is recommended to perform the Vargha and Delaney statistics (A_{12}), which is a non-parametric effect size measure for comparing which algorithm can have a higher chance of obtaining better results.
17		When comparing the difference significance of several multi-objective search algorithms, it is recommended to first perform the Kruskal–Wallis test together with the Bonferroni Correction. If the adjusted p-value is less than 0.05, the Shapiro-Wilk test is recommended to determine the normality of obtained data samples. If the data samples are normally distributed, *t*-test (parametric test) is recommended. Otherwise, the Mann-Whitney U test (non-parametric test) is recommended.
18		It is recommended to perform the Spearman's rank correlation coefficient (ρ) to measure the correlations between the obtained results and the complexity of problems when assessing the scalability of multi-objective search algorithms.

4 THE STATE OF ART

This section presents the state of the art related to uncertainty-wise testing for CPSs, which includes: 1) a summary of the primary studies collected in a systematic mapping study that we conducted to review testing of CPSs under uncertainty/non-determinism (Section 4.1) and a summary of the existing literature in terms of multi-objective test optimization (Section 4.2).

4.1 Summary of the Systematic Mapping

We conducted a systematic mapping study on testing CPSs under uncertainty by collecting and analyzing in total 26 research papers from the existing literature. In this section, we present the state-of-the-art on uncertainty-wise testing for CPSs.

The work reported in [115] deals with time simulation methods that are applicable for testing protocol software embedded in communicating systems. Two types of non-determinism are discussed in [115], which include: 1) internal non-determinism that refers to *"the nondeterministic implementation of the software under test"* [116-118]; and 2) external non-determinism that refers to *"is triggered by the concurrent (and reactive) environment of the software"* [119]. Internal non-determinism can be handled through changing the implementation of the SUT to remove non-determinism and test a set of serialization variants of the original implementation. To tackle external non-determinism, simulation time is used to control the level of non-determinism caused by the environment of the SUT. The test system is controlled in such a manner that external non-determinism is allowed, reduced or eliminated based on the instructions given by testers. To be more specific, external non-determinism can be eliminated through simulation scheduling that serializes the communication between the test environment and the SUT. Simulation with deterministic discrete timing is used to enhance the controllability of testing while allowing several possible orders for the events. By allowing external non-determinism, simulation time is used to scale timing in such a way that the tests can be executed in a host. The described methods have been evaluated in and applied to testing protocols for several standards.

In order to support black-box MBT of real-time embedded systems, the authors in [120] proposed a UML-based methodology to model the structure, behaviors, and constraints of the environment, where non-determinism was intensively captured. To be more specific, intervals were applied to model timeout transitions. The unexpected behaviors of users could be captured through non-deterministic attributes whose legal values were constrained in OCL. For the non-determinism of transitions, a probability distribution could be used. A set of stereotypes was proposed to capture the above-mentioned non-determinism. The approach was applied to and assessed using two industrial case studies. Based on the environment models, the black-box model based testing could be fully automated using search-based test case generation techniques and the generation of code simulating the environment.

The work in [121] extended the environment modeling methodology presented in [120] with a focus on the simulation for automated testing. A Java-based simulator was generated from the environment models using the model to text transformations. The non-determinism was handled when generating the simulator by a non-deterministic engine that obtained the values of non-deterministic occurrences from the simulation configuration generated by the test framework. Therefore, the simulation becomes deterministic for each specific simulation configuration. The test framework aimed to find a simulation configuration that could lead to a system failure/error using search-based techniques [122, 123]. Once such simulation configuration was found, a JUnit test case would be automatically generated, which is used to define two important components of the simulation including: (1) the configuration of the environment, e.g., number of sensors/actuators and their initialization; (2) the non-deterministic events in the simulation, e.g., variance in time-related events such as physical movements of hardware components, occurrence and type of hardware failures and actions of the user(s). The proposed techniques were evaluated with three artificial problems and two industrial real-time embedded systems.

Using input from the environment models [120], Iqbal et al. focused on the selection of the test cases to facilitate the black-box testing of real-time embedded systems and investigated three test automation strategies, i.e., random testing (as a baseline), adaptive random testing and genetic search-based testing in [124]. By random testing, test cases were randomly selected based on uniform probability. The basic idea of adaptive random testing was to reward diversity among test cases, as failing test cases usually tend to be clustered in contiguous regions of the input domain. A fitness function was defined to guide the search by estimating how close a test case can trigger a failure. The three approaches were validated using an industrial real-time embedded system, and the results showed that none of the three test automation strategies could fully dominate the others in all testing conditions. Moreover, the authors also provided practical guidelines in terms of applying these three techniques based on the experiment results.

In [125], a tool named UPPAAL-TRON [126] was proposed for online black-box conformance testing of real-time embedded systems based on non-deterministic timed automata specifications where non-deterministic action and timing were allowed. The notion of relativized timed input/output conformance was defined and it was possible to test the conformance of the system online based on the timed automata model and its environment assumptions that were explicitly captured. UPPAAL-TRON was implemented with a randomized online testing algorithm by extending the mature UPPAAL model-checker. It could generate and execute tests event-by-event in real-time through stimulating and monitoring the IUT (implementation under test). To perform these two functions, UPPAAL-TRON essentially computed the possible set of states symbolically based on the timed trace observed so far. The approach was further applied to test a rail-road intersection controller and was assessed in terms of error detection capability and computation performance. Moreover, some experience were shared in [126] with respect to applying UPPAAL-TRON in advanced electronic thermostat regulator that controls and monitors the temperature of industrial cooling plants.

Due to the lack of tool and techniques that could systematically tackle models capturing the indeterminacy as a result of concurrency, timing and limited observability and controllability, David et al. [127] proposed a number of principles and algorithms of model-based test generation with UPPAAL tool support, aiming at efficiently testing the real-time embedded system under uncertainty. The methods proposed were complementary with each other in terms of the allowed uncertainty and observability.

Specifically, online testing is effective when the system is highly non-deterministic [125, 126]. The main idea of online testing is to continually compute the possible set of states the model can occupy as test inputs/outputs or delays are observed. The uncertainty that the tester has about the possible state was thus captured by the state-set. While testing a system with partial observability, testing based on partially observable games techniques, such as UPPAAL-POTIGA, a timed game solver for partially observable timed games [128] could be applied.

So as to devise deterministic test requirements yielding the maximum network stress test scenarios for testing real-time embedded systems, the precise timing information of the interactions, i.e., messages in a UML sequence diagram, was required [129]. However, such information was not always available, and thus timing uncertainty would impact the effectiveness of the generated test cases in terms of revealing real-time faults. In order to address timing uncertainty when generating test requirements, Garousi adapted the barrier scheduling heuristic and proposed a wait-notify stress test methodology (WNSTM) [129]. To increase the chances of discovering real-time faults originating from network traffic overloads, WNSTM generated stress test requirements by installing barriers via a counting semaphore before the maximum-stressing messages of each sequence diagram. WNSTM was further evaluated on a prototype distributed real-time system based on a real-world case study, and the results showed that WNSTM was effective in terms of detecting real-time faults.

Ali et al. tackled the test minimization problem in the context of uncertainty-wise testing [130]. The heuristics were defined to minimize cost (time) and maximize effectiveness (transition coverage) and uncertainty-related attributes that were measured using uncertainty theory. The goal of this work was to evaluate the impact of various mutation and crossover operators on the performance of NSGA-II in terms of solving the test minimization problem. Three mutation operators and three crossover operators, resulting in nine combinations for the setting of NSGA-II, were evaluated in a real CPS case study. The results showed that the Blend Alpha crossover operator together with the polynomial mutation operator showed the best performance.

Executable test cases could be generated from test ready models yielding to subjective uncertainties, as they were developed based on testers' assumptions about the expected behaviors of a CPS, its expected physical environment, and the potential future deployments. Zhang et.al presented [36] a model evolution framework (*UncerTolve*) to interactively improve the quality of test ready models, i.e., reduce the uncertainty, based on operational data. *UncerTolve* was characterized by three key features: 1) model execution was applied to validate the syntactic correctness and conformance of test ready models against real operational data; 2) the objective uncertainty measurements of test ready models, i.e., probabilities, were evolved via model execution; and 3) a machine learning technique was applied to evolve state invariants and guards of transitions. *UncerTolve* was evaluated using one industrial CPS case study and the results were promising.

A model-based testing framework for probabilistic systems was presented in [131], where both an offline algorithm and an online algorithm for test generation were proposed. The probabilistic input/output transition systems (pIOTSs) were employed to capture the probabilistic requirements of the SUT, and statistical methods were applied to assess if the frequencies of the observation during test execution conformed to the probabilities in the requirements according to the conformance relation defined in this paper. This framework was proved to be mathematically correct via the classical soundness and completeness properties.

As an extension to their previous work in [131], the authors presented key concepts of an MBT framework for probabilistic systems with continuous time in [132]. A solid core of a probabilistic test theory was provided to handle real-time stochastic behaviors, which were captured via Markov automata. Compared with their previous work, the novelty of this work was the inclusion of stochastic time and exponential delays.

A smart device is non-deterministic in nature due to many reasons, such as the inaccuracy in an analog measurement. In order to assess the reliability confidence in the smart device under non-determinism, a black-box test environment was developed to automate the generation and execution of test data and the interpretation of the results in [82]. A finite state machine could be used to capture the system behavior, which was defined based on the behavior specified in the user manual. The non-determinism of the smart device response was addressed by a result checker.

In order to assert the correctness of CPS with respect to extra-functional properties, a conformance testing approach, Timed Testing of a physical Quantity (TTQ), was proposed in [133]. TTQ checked the conformance of a running CPS via the timed model checker Uppaal and the timed online testing tool TRON, with respect to a formal timed automata description utilizing measurements of physical quantities subject to uncertainties. The physical measurements obtained from a running CPS would be aggregated and translated into a linear observer trace model (TM). TM and the formal description of the system behaviors, i.e., timed automata, were then jointly executed to check the reachability of the terminal location of the trace model for inferring the conformance of the measurements with the expected behaviors of the system. In TTQ, the uncertainty was tackled in the sense that different intervals were employed for individual system modes of the hardware components. The proposed approach was evaluated for testing a wireless sensor node implementation with power measurements and the effectiveness of the approach was highlighted by the experiments.

Conformance testing of CPS yielded to the state space explosion problem, as it depended on a reachability check that required a state space traversal based on the TM and timed automata. Therefore, Woehrle et al. proposed a segmented state space traversal approach for conformance testing of CPS in their later research [134]. This approach could improve the scalability of quantitative conformance testing of a CPS and it was demonstrated based on a case study of two communicating sensor nodes.

A test environment was presented in [135] in order to achieve: 1) easy management of test cases and 2) automatic testing. A repository was devised to manage system functions to be tested and system configurations to be tested separately. The test environment could automatically generate test cases with a template engine from certain system functions in PROMELA (PROcess Meta LAnguage) and system configurations in XML for the device under test. A modified SPIN (Simple PROMELA Interpreter) model checker was applied to check certain properties of the PROMELA model where non-deterministic sequences were allowed. A prototype of an air conditioning system was developed for evaluating the proposed approach.

In the context of testing CPS product lines, authors in [136] proposed a methodology with tool support (ASTERYSCO) to automatically generate simulation-based test system instances for testing individual configurations of CPSs. A test system to be generated was described in Simulink encapsulating several sources, e.g., test cases or test oracles. Feature model was applied to manage the variability of a CPS as well as the test system. The uncertainty was dealt by simulating the physical world with the context environment. Moreover, reactive test cases were supported so that they could monitor and react to the various states of the CPS.

CPS is stochastic in nature due to actuator inaccuracies, sensor readings, the rate of arrivals, component failure rates, etc. which could affect the functional correctness of a CPS. In order to detect system operating conditions yielding the worst system robustness, Abbas et al. introduced a robustness-guided testing for verifying stochastic CPS in [137], which could quantify how robustly a stochastic CPS satisfied a specification in Metric Temporal Logic (MTL). The goal was achieved by transforming the testing problem to an expected robustness minimization problem that was solved with stochastic optimization algorithms, i.e., Markov chain Monte Carlo algorithms.

In order to test embedded systems interacting with a continuous environment, i.e., hybrid systems, a model-based mutation testing approach was proposed in [138]. The discrete controller behaviors were modeled by classical action systems whereas the evolutions of the environment were captured with qualitative differential equations. Non-determinism was allowed in the action system model, such as internal actions and non-deterministic updates. The basic idea behind mutation testing was to mutate the system models and generate test cases capable of killing a set of mutated models. The generated test cases were then applied to execute the SUT and check the conformance between the mutations and SUT.

In order to test the safety properties in a CPS, the authors in [139] provided a formal framework for conformance testing based on a formal conformance relation to guarantee transference of safety properties. The conformance relation was named as each set conformance, which was a weaker relation than the existing trace conformance. Hybrid automata were applied to formally model the system behaviors. The conformance testing was based on computations and over-approximations of the reachable set, which also took care of bounded errors of simulations or real measurements. Moreover, a model-based input selection algorithm based on a reach set coverage measure was presented in order to reduce the number of tests for a given set of test cases. The framework was evaluated in the domain of autonomous driving and the results showed that the conformance testing method could falsify more relations than the existing approaches.

A formal framework was provided in [140] for conformance testing of hybrid systems, a widely accepted mathematical model for many applications in embedded systems and CPSs. The goal of the conformance testing was to assert the conformance relation between the SUT and a formal specification using hybrid automata where non-determinism was allowed in both continuous and discrete dynamics. Moreover, based on the notion of star discrepancy, a coverage measure was proposed for hybrid systems in order to: 1) quantify the validation completeness and 2) guide input stimulus generation. A test generation method (named as gRRT) with a prototype tool support was then proposed based on rapidly exploring random tree algorithm for robotic planning guided by the coverage measure. gRRT was applied to a number of analog circuits and control applications. This test generation algorithm was later refined in [141] guided by both the coverage measure and the property to verify. In addition, the partial observability problem in test execution was also addressed in [141] through an estimation of the current location and the continuous state of the SUT with a hybrid observer.

Self-adaptation aims to address the challenges in CPSs capable of changing its behavior as well as structure in response to changes in the operating environment/context. Testing such self-adaptive behaviors is challenging due to the infinite reaction loop and uncertain interaction. A novel approach named as SIT (Sample-based Interactive Testing) was presented in [142] to test self-adaptive apps in an efficient and light-weight way. The input space of a self-adaptive app could be systematically split, adaptively explored and mapped to the testing of different behaviors by an interactive app model and a test generation technique. The impact of environmental constraints and uncertainty on an app's input/output pairs was captured in the interactive app model, enabling a systematic and guided exploration of its space. The test generation techniques only sampled inputs required by the exploration. Moreover, external uncertainty due to the errors in measuring environmental conditions was also handled in the SIT approach. An uncertainty specification was defined based on a set of functions mapping a given environment's output parameter to its corresponding app's input parameters together with the associated error range. The effectiveness and efficiency of SIT were evaluated with real-world self-adaptive apps.

Ramirez et al. proposed a search-based approach named as Loki to automatically explore environmental conditions capable of revealing requirements violations and latent behaviors in a dynamic adaptive system [143]. To be more specific, Loki first generated configurations specifying the type, duration, and severity of noise for each sensor in a dynamic adaptive system. Secondly, Loki simulated the system and environmental conditions according to the generated configurations. During simulation, a set of utility functions was applied to assess the satisfaction of the system requirement captured with goal based models and guide the search towards those environmental conditions producing the most distinct behaviors. Loki was applied to an autonomous vehicle system to illustrate several examples of requirement violations and latent behaviors.

Operating contexts for dynamic adaptive systems constitute main uncertainty that may affect the system behaviors at a run time. An evolutionary computation-based approach (FENRIR) was proposed in [144]. FENRIR could automatically explore how the varying operational contexts affected a dynamic adaptive system with instrumented code by searching for the system and environmental parameters that could produce previously unexamined system execution traces. The resulting execution traces could facilitate the identification of potential bugs in the code. FENRIR was evaluated on an industry-provided problem, management of a remote data mirroring network. Experiment results showed that FENRIR could provide a significantly greater coverage of execution paths as compared with randomized testing. Moreover, in their following research, Fredericks et al. [145] further proposed an evolutionary computation-based approach (named Veritas) to adapt requirement-based test cases at runtime in response to different system and environmental conditions. Specifically, Veritas monitored the environment to collect evidence of changes in the environment and adapted individual test cases to ensure testing relevance that was assessed with utility functions. An online evolutionary algorithm (i.e., (1+1)-ONLINE EA) was then applied to facilitate run-time adaptation of test case parameters once required. Veritas was evaluated using an intelligent robotic vacuum, and the results showed that Veritas could largely reduce the mismatch between test cases and operational context caused by uncertainty within the system and the environment.

4.2 Multi-objective Test Optimization

Many software testing problems are multi-objective in nature (e.g., test set minimization problem [84, 111]) and the existing literature mainly studied the multi-objective test optimization from three perspectives, i.e., test case selection, test set minimization and test case prioritization [83, 108, 110, 146-148]. *Test Case Selection* aims at selecting a set of relevant test cases from the entire test set for cost-effectively testing the systems or programs under test. *Test Set Minimization* minimizes a given test set for eliminating redundant test cases with the aim to reduce the total cost of testing (e.g., execution time) while preserving high effectiveness (e.g., fault detection capability). *Test Case Prioritization* prioritizes a test set into an optimal order for detecting faults as early as possible while satisfying other pre-defined criteria, e.g., reducing the execution time of test cases. It is worth mentioning that it is practically infeasible to only take a single objective into account when studying the above-mentioned three testing problems [146]. Thus, researchers usually focus on proposing, defining and formulating a set of cost-effective objectives based on specific contexts and domains (e.g., telecommunication [84, 111]), which are further taken as input for guiding the search towards obtaining trade-off solutions that can balance different criteria [72, 83, 105, 106, 108, 110, 146, 149, 150].

The principle foundation of Search-Based Software Testing (SBST) is to formulate and encode various multi-objective software testing problems into multi-objective test optimization problems that can be efficiently solved using various multi-objective search algorithms (e.g., non-dominated sorting genetic algorithm II (NSGA-II) [73]). According to a publicly-available SBSE repository maintained by the CREST center [151], in total 821 research papers have been focusing on tackling a variety of testing challenges and out of these 821 works, a large number of multi-objective test problems have been increasingly researched, e.g., test case selection problem [105, 110] and test case prioritization [72, 106, 108]. Harman et al. have conducted a systematic literature review for SBSE [149] that described and sketched a variety of SE applications employing search algorithms to address multi-objective test optimization problems. In particular, a set of objectives have been listed in [146] (e.g., fault detection capability, code coverage), which can be used for multi-objective test optimization in the context of software testing and have been researched in the state-of-the-art [83].

For instance, in terms of test case selection, two cost-effective objectives (i.e., code coverage, execution time) have been defined with the aim of selecting optimal test cases in [152]. The two-objective problem has been further converted into a single-objective problem by assigning weights to each objective followed by solving the problem using search algorithms [152]. Yoo and Harman [110] formally defined three cost-effectiveness measures (i.e., code coverage, fault detection history, and execution time) in the context of regression testing and employed a greedy algorithm and a multi-objective search algorithm NSGA-II for selecting optimal test cases from a given test set. Our previous work [105] proposed to consider the importance of test cases, the potential impact caused by test case failure and the likelihood of detecting faults by test cases into account when performing test case selection. We further empirically evaluated eight existing multi-objective search algorithms (e.g., NSGA-II) with two weight assignment strategies (i.e., fixed-weight strategy and randomly-assigned weight strategy) using one industrial case study and a large number of artificial problems. With respect to test set minimization, our previous work [84] attempted to deal with minimization problem by defining five cost-effectiveness measures (e.g., test minimization percentage, feature pairwise coverage and fault detection capability). We further empirically evaluated the performance of eight multi-objective search algorithms (including NSGA-II and RWGA) in conjunction with our defined fitness function using one industrial case study and 500 artificial problems. An improved search algorithm was proposed in [108] with the aim of incorporating predefined user preferences (based on different domains) when solving multi-objective test optimization problems (e.g., test set minimization problem). As for test case prioritization, a time-aware test case prioritization problem has been addressed in [153] by employing linear integer programming with the aim of prioritizing test cases into an optimal order with a given time budget. Our previous work [106] studied the test case prioritization problem by proposing four effectiveness measures (e.g., configuration coverage, test API coverage, fault detection capability) in the context of telecommunication domain. Two prioritization strategies were further defined and incorporated into the fitness function, which include: 1) *Incremental Unique Coverage* that only considers the incremental unique elements (e.g., test APIs) covered by the test case as compared with the ones covered by the prioritized test cases; and 2) *Position Impact* that gives more impact to a test case with a higher execution position (i.e., scheduled to be executed earlier).

However, to the best of our knowledge, there are only a few works [33, 78] in the literature (as discussed in Section 4.1) that addressed the uncertainty-wise multi-objective test optimization problem that explicitly takes uncertainty into account. Thus, there is still a large open research room that requires further investigation in terms of this angle.

5 CONCLUSION AND FUTURE RESEARCH DIRECTIONS

Due to the fact that uncertainty is unavoidable in the behaviors of Cyber-Physical Systems (CPSs), it is important that uncertainty in CPSs both in its internal behavior and its physical environment must be considered explicitly during their verification and validation. In this direction, this chapter introduced uncertainty-wise testing in the context of Cyber-Physical Systems (CPSs). More specifically, we summarized our existing works from the angles of Uncertainty-wise Model-Based Testing, Uncertainty-wise Test Ready Model Evolution, and Uncertainty-wise Multi-Objective Test Optimization, in addition to providing potential future research directions. Moreover, we also provided a survey of the existing uncertainty-wise testing approaches to present where the current state-of-the-art stands in this area.

Given the anticipated substantial rise in CPS applications in the future in both daily lives and critical domains, it will become even more critical for CPSs to handle uncertainty in a reliable way. This opens up a lot of new research directions for uncertainty-wise testing in the future. We summarize some research directions in the area of uncertainty-wise testing below:

First, existing uncertainty-wise testing solutions support only testing functional behaviors in the presence of environment uncertainty with a limited extent as we discussed in the related work section. In this direction, new research is required to define novel testing methods for testing extra-functional behaviors of CPSs, e.g., related to security and safety, under various types of uncertainties. Such testing methods will ensure that extra-functional requirements (e.g., safety and security requirements) are not violated under various types of uncertainties. Note that if extra-functional requirements are not violated in nominal conditions, it doesn't mean that such requirements will not be violated under uncertainty and thus it is important that future research in uncertainty-wise testing shall also focus on testing extra-functional requirements in the presence of uncertainty.

Second, due to the fact that a large number of CPSs interact with humans, testing methods must take into consideration uncertainty introduced by humans into CPS operations. Given that understanding human behavior requires expertise from the domain of human psychology, such testing methods shall incorporate knowledge from the domain of human psychology to be considered explicitly in all the phases of testing. In addition to uncertainty related theories, such as Uncertainty and Probability theories, testing solutions shall also be grounded on theories related to study of human behaviors in uncertainty such as Prospect theory [154]. Needless to say, such testing solutions shall be interdisciplinary requiring expertise from the domains of ICT and Human Psychology with the eventual implementations as novel testing tool suites.

Third, the future modeling solutions to support automated testing shall provide integrated solutions incorporating modeling expected behavior of software, hardware, and physical environment including humans. In addition, the modeling solutions must provide an implementation of relevant uncertainty related theories, e.g., Uncertainty Theory and Probability Theory, and theories from other domains such as Prospect Theory from human psychology.

Fourth, the current modeling tools for system design and also for model-based testing only support checking the syntactic correctness of models; however, none of the existing modeling tools support automated validation of test models. This opens up a new research direction for devising new methods to validate test ready models of CPSs capturing uncertainty. In terms of testing, it is important that test ready models are semantically correct; otherwise, test cases generated from such test ready models will not be correct. In the presence of uncertainty, such validation methods become even more complicated because of lack of knowledge or missing information as compared to the situation where test ready models do not capture uncertainty.

Fifth, novel test strategies for uncertainty-wise testing of CPSs are required to be developed in the future. Such test strategies shall not only focus on finding optimized test paths in test ready models but shall also focus on finding best test data including uncertainty test data, to find faults cost-effectively. Notice that with uncertainty-wise test strategies, we are looking for faults that could only be observed in the presence of uncertainty, thus requiring the development of a potentially new classification of uncertainty-wise faults for CPSs. Such a classification can further facilitate the development of new mutation operators that can be used to assess the cost-effectiveness of newly defined uncertainty-wise test generation strategies and support building the foundations of mutation testing for uncertainty-wise testing. Finally, in terms of test optimization, we forsee the need for the development of new search algorithms for both single objective and multi-objective optimization such as extensions to Genetic Algorithms (GAs) and Non-Dominated Sorting Genetic Algorithm II (NSGA-II). Such algorithms shall be extended to incorporate uncertainty in their inputs to provide optimized solutions even when faced with uncertain inputs.

Sixth, as with any area of research, more empirical evaluations are required for uncertainty-wise testing. This includes not only the empirical evaluations in academic settings but also more applications in diverse domains to determine which uncertainty-wise techniques are better under which contexts.

Finally, as we discussed that there is no existing standard for modeling uncertainty and recently a new Uncertainty Modeling [29] standard is initiated at the Object Management Group (OMG). This standard will be a generic uncertainty modeling standard and will probably require tailoring for specialized domains and applications. In addition, more standardization efforts are expected to integrate uncertainty related aspects into existing software and systems modeling languages such as SysML [55] and MARTE [56], in addition to testing standards, such as the UML Testing Profile [53].

6 ACKNOWLEDGMENT

This research was supported by the EU Horizon 2020 funded project (Testing Cyber-Physical Systems under Uncertainty, Project Number: 645463). Tao Yue and Shaukat Ali are also supported by RCN funded Zen-Configurator project, RFF Hovedstaden funded MBE-CR project, RCN funded MBT4CPS project, RCN funded Certus SFI and the EU COST action MPM4CPS. Hong Lu is funded by RCN funded Zen-Configurator project, Shuai Wang is funded by RFF Hovedstaden funded MBE-CR project and RCN funded Certus SFI, and Man Zhang is funded by the EU Horizon 2020 funded project (Testing Cyber-Physical Systems under Uncertainty, Project Number: 645463).

References

[1] D. B. Rawat, J. J. Rodrigues, and I. Stojmenovic, Cyber-physical systems: from theory to practice, CRC Press, 2015.

[2] S. Sunder, Foundations for Innovation in Cyber-Physical Systems, in Proceedings of the NIST CPS Workshop, Chicago, IL, USA, 2012.

[3] E. Geisberger, and M. Broy, Living in a networked world: Integrated research agenda Cyber-Physical Systems (agendaCPS), Herbert Utz Verlag, 2015.

[4] S. Hangal, and M. S. Lam, Tracking down software bugs using automatic anomaly detection, in Proceedings of the 24th International Conference on Software Engineering (ICSE 2002). pp. 291-301, 2002.

[5] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn, A. Bertolino, J. Jenny Li, and H. Zhu, An orchestrated survey of methodologies for automated software test case generation, Journal of Systems and Software, vol. 86, no. 8 (2013) 1978-2001, http://dx.doi.org/10.1016/j.jss.2013.02.061.

[6] C. Nie, and H. Leung, A survey of combinatorial testing, ACM Computing Surveys (CSUR), vol. 43, no. 2 (2011) 11.

[7] P. McMinn, Search-based software test data generation: A survey, Software Testing Verification and Reliability, vol. 14, no. 2 (2004) 105-156.

[8] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, A systematic review of the application and empirical investigation of search-based test case generation, IEEE Transactions on Software Engineering, vol. 36, no. 6 (2010) 742-762.

[9] J. Burnim, and K. Sen, Heuristics for scalable dynamic test generation, in 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2008. pp. 443-446, 2008.

[10] L. Cseppento, and Z. Micskei, Evaluating symbolic execution-based test tools, in 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST). pp. 1-10, 2015.

[11] J. Offutt, and A. Abdurazik, Generating tests from UML specifications, in International Conference on the Unified Modeling Language. pp. 416-429, 1999.

[12] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann, Generating test data from state-based specifications, Software testing, verification and reliability, vol. 13, no. 1 (2003) 25-53.

[13] M. Utting, A. Pretschner, and B. Legeard, A taxonomy of model-based testing approaches, Software Testing, Verification and Reliability, vol. 22, no. 5 (2012) 297-312.

[14] M. Harman, and P. McMinn, A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search, IEEE Transactions on Software Engineering, vol. 36, no. 2 (2010) 226-247, 10.1109/TSE.2009.71.

[15] M. Utting, and B. Legeard, Practical model-based testing: a tools approach, San Francisco, CA, USA, Morgan Kaufmann, 2010.

[16] T. Yue, S. Ali, and M. Zhang, RTCM: A Natural Language Based, Automated, and Practical Test Case Generation Framework, in Proceedings of the 2015 International Symposium on Software Testing and Analysis. pp. 397-408, 2015.

[17] S. Ali, M. Zohaib Iqbal, A. Arcuri, and L. C. Briand, Generating test data from OCL constraints with search techniques, Software Engineering, IEEE Transactions on, vol. 39, no. 10 (2013) 1376-1402.

[18] M. Mitchell, An Introduction to Genetic Algorithms, Cambridge, The MIT Press, 1996.

[19] "Evosuite–automatic test suite generation for java," accessed 2017; http://www.evosuite.org/.

[20] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, Understanding Uncertainty in Cyber-Physical Systems: A Conceptual Model, in Proceedings of the 12th European Conference on Modelling Foundations and Applications (ECMFA). pp. 247-264, 2016.

[21] M. Zhang, S. Ali, T. Yue, and R. Norgre, An Integrated Modeling Framework to Facilitate Model-Based Testing of Cyber-Physical Systems under Uncertainty, Technical report 2016-02, Simula Research Laboratory, 2016; https://www.simula.no/publications/integrated-modeling-framework-facilitate-model-based-testing-cyber-physical-systems.

[22] M. Zhang, T. Yue, S. Ali, B. Selic, O. Okariz, R. Norgren, and K. Intxausti, Specifying Uncertainty in Use Case Models in Industrial Settings, Technical Report 2016-15, Simula Research Laboratory, 2016; https://www.simula.no/publications/specifying-uncertainty-use-case-models-industrial-settings.

[23] "U-RUCM: Specifying Uncertainty in Use Case Models," accessed; http://zen-tools.com/rucm/U_RUCM.html.
[24] H. Zhang, T. Yue, S. Ali, and C. Liu, Facilitating Requirements Inspection with Search-Based Selection of Diverse Use Case Scenarios, in BICT'15 Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS). pp. 229-236, 2015.
[25] T. Yue, L. C. Briand, and Y. Labiche, aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 24, no. 3 (2015) 13.
[26] T. Yue, L. C. Briand, and Y. Labiche, Facilitating the Transition From Use Case Models to Analysis Models: Approach and Experiments, ACM Transactions on Software Engineering and Methodology, vol. 22, no. 1 (2013).
[27] T. Weilkiens, Systems engineering with SysML/UML: modeling, analysis, design, Morgan Kaufmann, 2011.
[28] S. Friedenthal, A. Moore, and R. Steiner, A practical guide to SysML: the systems modeling language, Morgan Kaufmann, 2014.
[29] "Uncertainty Modeling (UM)," accessed; http://www.omgwiki.org/uncertainty/doku.php?id=start.
[30] H. Song, D. B. Rawat, S. Jeschke, and C. Brecher, Cyber-Physical Systems: Foundations, Principles and Applications, Morgan Kaufmann, 2016.
[31] C. Talcott, Cyber-Physical Systems and Events, Software-Intensive Systems and New Computing Paradigms: Challenges and Visions, M. Wirsing, J.-P. Banâtre, M. Hölzl and A. Rauschmayer, eds., pp. 101-115, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.
[32] S. Ali, L. C. Briand, M. J.-u. Rehman, H. Asghar, M. Z. Z. Iqbal, and A. Nadeem, A state-based approach to integration testing based on UML models, Inf. Softw. Technol., vol. 49, no. 11-12 (2007) 1087-1106, 10.1016/j.infsof.2006.11.002.
[33] M. Zhang, S. Ali, T. Yue, and M. Hedman, Uncertainty-based Test Case Generation and Minimization for Cyber-Physical Systems: A Multi-Objective Search-based Approach, Technical report 2016-13, Simula Research Laboratory, 2016; https://www.simula.no/publications/uncertainty-based-test-case-generation-and-minimization-cyber-physical-systems-multi.
[34] L. Briand, and Y. Labiche, A UML-based approach to system testing, in International Conference on the Unified Modeling Language. pp. 194-208, 2001.
[35] A. Abdurazik, and J. Offutt, Using UML collaboration diagrams for static checking and test generation, in International Conference on the Unified Modeling Language. pp. 383-395, 2000.
[36] M. Zhang, S. Ali, T. Yue, and R. Norgre, Uncertainty-wise evolution of test ready models, Information and Software Technology (2017), http://dx.doi.org/10.1016/j.infsof.2017.03.003.
[37] M. Zhang, S. Ali, T. Yue, and R. Norgren, Interactively Evolving Test Ready Models with Uncertainty Developed for Testing Cyber-Physical Systems, Technical Report 2016-12, Simula Research Laboratory, 2016; https://www.simula.no/publications/interactively-evolving-test-ready-models-uncertainty-developed-testing-cyber-physical.
[38] P. Samuel, R. Mall, and A. K. Bothra, Automatic test case generation using unified modeling language (UML) state diagrams, IET software, vol. 2, no. 2 (2008) 79-93.
[39] L. C. Briand, Y. Labiche, and Y. Wang, Using simulation to empirically investigate test coverage criteria based on statechart. pp. 86-95, 2004.
[40] W. Feller, An introduction to probability theory and its applications, John Wiley & Sons, 2008.
[41] B. Liu, Uncertainty theory, Springer, 2015.
[42] C. C. Michael, G. McGraw, and M. A. Schatz, Generating software test data by evolution, IEEE transactions on software engineering, vol. 27, no. 12 (2001) 1085-1110.
[43] A. J. Offutt, and J. Pan, Automatically detecting equivalent mutants and infeasible paths, Software testing, verification and reliability, vol. 7, no. 3 (1997) 165-192.
[44] M. Harman, K. Lakhotia, and P. McMinn, A multi-objective approach to search-based test data generation, in Proceedings of the 9th annual conference on Genetic and evolutionary computation. pp. 1098-1105, 2007.
[45] S. Ali, M. Z. Iqbal, A. Arcuri, and L. C. Briand, Generating test data from OCL constraints with search techniques, IEEE Transactions on Software Engineering, vol. 39, no. 10 (2013) 1376-1402.
[46] "GraphWalker," accessed 2017; http://graphwalker.github.io/.

[47] "MoMuT," accessed 2017; https://www.momut.org/.
[48] "OSMO MBT Tool," accessed 2017; https://github.com/mukatee/osmo.
[49] "TestCast MBT - Model Based Testing Platform," accessed; http://www.elvior.com/en.
[50] "Conformiq Creator&Designer," accessed 2017; https://www.conformiq.com/products/.
[51] "MBTsuite - The Testing Framework," accessed; http://www.mbtsuite.com/home.html.
[52] "National Standards Bodies," accessed 2015; http://www.nist.gov/iaao/stnd-org.cfm.
[53] OMG, UML Testing Profile, 2013, http://www.omg.org/spec/UTP/.
[54] OMG, Unified Modeling Language (UML), 2015, http://www.omg.org/spec/UML/.
[55] OMG, System Modeling Language, 2014, http://www.omg.org/spec/SysML/.
[56] OMG, UML Profile For MARTE: Modeling And Analysis Of Real-Time Embedded Systems, 2011, http://www.omg.org/spec/MARTE/.
[57] OMG, Object Constraint Language (OCL), 2014, http://www.omg.org/spec/OCL/.
[58] OMG, "Meta Object Facility (MOF) Core Specification (Version 2.4.2)," 2014, http://www.omg.org/spec/MOF/2.4.2.
[59] ISO/IEC, "ISO/IEC 19505-x: Information technology - Object Management Group Unified Modeling Language (OMG UML)," 2012.
[60] ISO/IEC, "ISO/IEC 19507: Information technology - Object Management Group Object Constraint Language (OCL)," 2012.
[61] ISO/IEC, "ISO/IEC 19506:2012 Information technology -- Object Management Group Architecture-Driven Modernization (ADM) -- Knowledge Discovery Meta-Model (KDM)," 2012, http://www.iso.org/iso/catalogue_detail.htm?csnumber=32625.
[62] ISO/IEC, "ISO/IEC 10746-x: Information technology - Open Distributed Processing," 2009.
[63] ETSI, "ETSI ES 202 951 Methods for Testing and Specification (MTS); Model-Based Testing (MBT); Requirements for Modelling Notations (v1,1,1)," 2011, http://www.etsi.org/deliver/etsi_es/202900_202999/202951/01.01.01_60/es_202951v010101p.pdf.
[64] ETSI, "ETSI EG 201 015 Methods for Testing and Specification (MTS); Standards engineering process; A Handbook of validation methods (v1.2.1)," 2012, http://www.etsi.org/deliver/etsi_eg/201000_201099/201015/02.01.01_60/eg_201015v020101p.pdf.
[65] ISO/IEC/IEEE, "ISO/IEC/IEEE 29119-x: Software Testing," 2013, http://www.softwaretestingstandard.org/.
[66] ISO/IEC, "ISO/IEC 9646-x:Information technology -- Open Systems Interconnection -- Conformance testing methodology and framework," 1994.
[67] IEEE, "IEEE Standard for System and Software Verification and Validation," IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004), 2012, pp. 1-223, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6204026.
[68] IEEE, "IEEE Standard for Software and System Test Documentation - Redline," IEEE Std 829-2008 (Revision of IEEE Std 829-1998) - Redline, 2008, pp. 1-161.
[69] IEEE, "IEEE Standard for Software Unit Testing," ANSI/IEEE Std 1008-1987, 1986, p. 0_1, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=27763.
[70] IEEE, "IEEE Standard Classification for Software Anomalies," IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993), 2010, pp. 1-23, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5399061.
[71] IEC, "IEC 61508: Commented version, Functional safety of electrical/electronic/programmable electronic safety-related systems," 2010, http://www.iec.ch/functionalsafety/standards/.
[72] S. Wang, S. Ali, T. Yue, Ø. Bakkeli, and M. Liaaen, Enhancing Test Case Prioritization in an Industrial Setting with Resource Awareness and Multi-Objective Search, in Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16). pp. 182-191, 2016.
[73] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, Evolutionary Computation, IEEE Transactions on, vol. 6, no. 2 (2002) 182-197, 10.1109/4235.996017.
[74] A. Arcuri, and L. 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, Waikiki, Honolulu, HI, USA, 2011, pp. 1-10.
[75] A. Arcuri, and G. Fraser, On Parameter Tuning in Search Based Software Engineering, in International Symposium on Search Based Software Engineering (SSBSE), 2011.

[76] S. Ali, and T. Yue, Evaluating Normalization Functions with Search Algorithms for Solving OCL Constraints, Testing Software and Systems: 26th IFIP WG 6.1 International Conference, ICTSS 2014, Madrid, Spain, September 23-25, 2014. Proceedings, M. G. Merayo and E. M. de Oca, eds., pp. 17-31, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.
[77] S. Ali, T. Yue, and M. Zhang, Tackling Uncertainty in Cyber-Physical Systems with Automated Testing, ADA User Journal, vol. 37, no. 4 (2016).
[78] S. Ali, Y. Li, T. Yue, and M. Zhang, Uncertainty-Wise and Time-Aware Test Case Prioritization with Multi-Objective Search, Technical report 2017-03, Simula Research Laboratory, 2017; https://www.simula.no/publications/uncertainty-wise-and-time-aware-test-case-prioritization-multi-objective-search.
[79] J. Black, E. Melachrinoudis, and D. Kaeli, Bi-criteria models for all-uses test suite reduction, in Proceedings of the 26th International Conference on Software Engineering. pp. 106-115, 2004.
[80] "jMetal," accessed 2016; http://jmetal.sourceforge.net/.
[81] "Future Position X," accessed 2017; http://www.fpx.se/.
[82] P. Bishop, and L. Cyra, Overcoming non-determinism in testing smart devices: a case study, in International Conference on Computer Safety, Reliability, and Security. pp. 237-250, 2010.
[83] S. Yoo, and M. Harman, Regression testing minimization, selection and prioritization: a survey, Softw. Test. Verif. Reliab., vol. 22, no. 2 (2012) 67-120, 10.1002/stv.430.
[84] S. Wang, S. Ali, and A. Gotlieb, Cost-effective test suite minimization in product lines using search techniques, Journal of Systems and Software (2014).
[85] S. Elbaum, A. G. Malishevsky, and G. Rothermel, Test case prioritization: a family of empirical studies, IEEE Transactions on Software Engineering, vol. 28, no. 2 (2002) 159-182, 10.1109/32.988497.
[86] M. Harder, J. Mellen, and M. D. Ernst, Improving test suites via operational abstraction, in Proceedings of the 25th international conference on Software engineering. pp. 60-71, 2003.
[87] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, Coverage-based regression test case selection, minimization and prioritization: a case study on an industrial system, Software Testing, Verification and Reliability, vol. 25, no. 4 (2015) 371-396.
[88] R. C. Bryce, C. J. Colbourn, and M. B. Cohen, A framework of greedy methods for constructing interaction test suites, in Proceedings of the 27th international conference on Software engineering. pp. 146-155, 2005.
[89] R. C. Bryce, and C. J. Colbourn, Test prioritization for pairwise interaction coverage. pp. 1-7, 2005.
[90] S. McMaster, and A. Memon, Call Stack Coverage for GUI Test-Suite Reduction, in 2006 17th International Symposium on Software Reliability Engineering. pp. 33-44, 2006.
[91] M. Marr, and A. Bertolino, Using Spanning Sets for Coverage Testing, IEEE Trans. Softw. Eng., vol. 29, no. 11 (2003) 974-984, 10.1109/tse.2003.1245299.
[92] D. Jeffrey, and N. Gupta, Test suite reduction with selective redundancy, in Software Maintenance, 2005 (ICSM'05). pp. 549-558, 2005.
[93] Z. Li, M. Harman, and R. M. Hierons, Search algorithms for regression test case prioritization, IEEE Transactions on software Engineering, vol. 33, no. 4 (2007).
[94] T. A. Budd, Mutation analysis of program test data, (1980).
[95] J. C. Munson, and S. G. Elbaum, Code churn: A measure for estimating the impact of code change, in Proceedings of 1998 International Conference on Software Maintenance. pp. 24-31, 1988.
[96] G. Rothermel, R. H. Untch, C. Chengyun, and M. J. Harrold, Prioritizing test cases for regression testing, IEEE Transactions on Software Engineering, vol. 27, no. 10 (2001) 929-948, 10.1109/32.962562.
[97] S. Elbaum, A. Malishevsky, and G. Rothermel, Incorporating varying test costs and fault severities into test case prioritization, in Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001. pp. 329-338, 2001.
[98] H. Srikanth, L. Williams, and J. Osborne, System test case prioritization of new and regression test cases, in 2005 International Symposium on Empirical Software Engineering, 2005. p. 10 pp., 2005.
[99] C. Hettiarachchi, H. Do, and B. Choi, Risk-based test case prioritization using a fuzzy expert system, Information and Software Technology, vol. 69 (2016) 1-15, 1//, http://dx.doi.org/10.1016/j.infsof.2015.08.008.
[100] T. Ma, H. Zeng, and X. Wang, Test case prioritization based on requirement correlations, in 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). pp. 419-424, 2016.

[101] R. Krishnamoorthi, and S. A. Sahaaya Arul Mary, Factor oriented requirement coverage based system test case prioritization of new and regression test cases, Information and Software Technology, vol. 51, no. 4 (2009) 799-808, http://dx.doi.org/10.1016/j.infsof.2008.08.007.

[102] S. Tahvili, S. Mehrdad, L. Stig, A. Wasif, B. Markus, and S. Daniel, Dynamic Integration Test Selection Based on Test Case Dependencies, in 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). pp. 277-286, 2016.

[103] B. Miranda, and A. Bertolino, Scope-aided test prioritization, selection and minimization for software reuse, Journal of Systems and Software (2016), http://dx.doi.org/10.1016/j.jss.2016.06.058.

[104] G. Fraser, and F. Wotawa, Property relevant software testing with model-checkers, ACM SIGSOFT Software Engineering Notes, vol. 31, no. 6 (2006) 1-10.

[105] D. Pradhan, S. Wang, S. Ali, and T. 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, Denver, Colorado, USA, 2016, pp. 1085-1092.

[106] D. Pradhan, S. Wang, S. Ali, T. Yue, and M. Liaaen, STIPI: Using Search to Prioritize Test Cases Based on Multi-objectives Derived from Industrial Practice, Testing Software and Systems: 28th IFIP WG 6.1 International Conference, ICTSS 2016, Graz, Austria, October 17-19, 2016, Proceedings, F. Wotawa, M. Nica and N. Kushik, eds., pp. 172-190, Cham: Springer International Publishing, 2016.

[107] S. Wang, S. Ali, T. Yue, Y. Li, and M. 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, Austin, Texas, 2016, pp. 631-642.

[108] S. Wang, S. Ali, T. Yue, and M. Liaaen, UPMOA: An improved search algorithm to support user-preference multi-objective optimization, in 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE). pp. 393-404, 2015.

[109] S. Wang, S. Ali, and A. Gotlieb, Random-Weighted Search-Based Multi-objective Optimization Revisited, Search-Based Software Engineering: 6th International Symposium, SSBSE 2014, Fortaleza, Brazil, August 26-29, 2014. Proceedings, C. Le Goues and S. Yoo, eds., pp. 199-214, Cham: Springer International Publishing, 2014.

[110] S. Yoo, and M. Harman, Pareto efficient multi-objective test case selection, in Proceedings of the 2007 international symposium on Software testing and analysis, London, United Kingdom, 2007, pp. 140-150.

[111] S. Wang, S. Ali, and A. Gotlieb, Minimizing test suites in software product lines using weight-based genetic algorithms, in Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference. pp. 1493-1500, 2013.

[112] G. Fliedl, W. Mayerthaler, C. Winkler, C. Kop, and H. C. Mayr, Enhancing requirements engineering by natural language based conceptual predesign, in IEEE SMC '99 pp. 778-783 vol.5, 1999.

[113] J. H. Mcdonald, Handbook of Biological Statistics, Baltimore, Maryland, U.S.A., Sparky House Publishing, 2009.

[114] D. J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, Chapman and Hall/CRC, 2007.

[115] J. Latvakoski, and H. Honka, Time simulation methods for testing protocol software embedded in communicating systems, Testing of Communicating Systems, pp. 379-394: Springer, 1999.

[116] R. H. Carver, and K.-C. Tai, Replay and testing for concurrent programs, IEEE Software, vol. 8, no. 2 (1991) 66-74.

[117] S. N. Weiss, A formal framework for the study of concurrent program testing, in Software Testing, Verification, and Analysis, 1988., Proceedings of the Second Workshop on. pp. 106-113, 1988.

[118] W. Schutz, A test strategy for the distributed real-time system MARS, in CompEuro'90. Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering. pp. 20-27, 1990.

[119] M. Kim, S. T. Chanson, and S. Yoo, Design for testability of protocols based on formal specifications, Protocol Test Systems VIII, pp. 252-264: Springer, 1996.

[120] M. Z. Iqbal, A. Arcuri, and L. Briand, Environment modeling with UML/MARTE to support black-box system testing for real-time embedded systems: methodology and industrial case studies, in International Conference on Model Driven Engineering Languages and Systems. pp. 286-300, 2010.

[121] M. Z. Iqbal, A. Arcuri, and L. Briand, Environment modeling and simulation for automated testing of soft real-time embedded software, Software & Systems Modeling, vol. 14, no. 1 (2015) 483-524.

[122] M. Z. Iqbal, A. Arcuri, and L. Briand, Combining search-based and adaptive random testing strategies for environment model-based testing of real-time embedded systems, in International Symposium on Search Based Software Engineering. pp. 136-151, 2012.

[123] M. Z. Iqbal, A. Arcuri, and L. Briand, Empirical investigation of search algorithms for environment model-based testing of real-time embedded software, in Proceedings of the 2012 International Symposium on Software Testing and Analysis. pp. 199-209, 2012.

[124] A. Arcuri, M. Z. Iqbal, and L. Briand, Black-box system testing of real-time embedded systems using random and search-based testing, in IFIP International Conference on Testing Software and Systems. pp. 95-110, 2010.

[125] K. G. Larsen, M. Mikucionis, and B. Nielsen, Online testing of real-time systems using uppaal, in International Workshop on Formal Approaches to Software Testing. pp. 79-94, 2004.

[126] K. G. Larsen, M. Mikucionis, B. Nielsen, and A. Skou, Testing real-time embedded software using UPPAAL-TRON: an industrial case study, in Proceedings of the 5th ACM international conference on Embedded software. pp. 299-306, 2005.

[127] A. David, K. G. Larsen, S. Li, M. Mikucionis, and B. Nielsen, Testing real-time systems under uncertainty, in International Symposium on Formal Methods for Components and Objects. pp. 352-371, 2010.

[128] A. David, K. G. Larsen, S. Li, and B. Nielsen, Timed testing under partial observability, in Software Testing Verification and Validation, 2009. ICST'09. International Conference on. pp. 61-70, 2009.

[129] V. Garousi, Traffic-aware stress testing of distributed real-time systems based on UML models in the presence of time uncertainty, in Software Testing, Verification, and Validation, 2008 1st International Conference on. pp. 92-101, 2008.

[130] S. Ali, Y. Li, T. Yue, and M. Zhang, An Empirical Evaluation of Mutation and Crossover Operators for Multi-Objective Uncertainty-Wise Test Minimization, in 10th International Workshop on Search-based Software Testing, 2017.

[131] M. Gerhold, and M. Stoelinga, Model-Based Testing of Probabilistic Systems, in International Conference on Fundamental Approaches to Software Engineering. pp. 251-268, 2016.

[132] M. Gerhold, and M. Stoelinga, Model-based testing of stochastic systems with IOCO theory, 2016.

[133] M. Woehrle, K. Lampka, and L. Thiele, Conformance testing for cyber-physical systems, ACM Transactions on Embedded Computing Systems (TECS), vol. 11, no. 4 (2012) 84.

[134] M. Woehrle, K. Lampka, and L. Thiele, Segmented state space traversal for conformance testing of cyber-physical systems, in International Conference on Formal Modeling and Analysis of Timed Systems. pp. 193-208, 2011.

[135] T. Kuroiwa, Y. Aoyama, and N. Kushiro, Testing Environment for CPS by Cooperating Model Checking with Execution Testing, Procedia Computer Science, vol. 96 (2016) 1341-1350.

[136] A. Arrieta, G. Sagardui, L. Etxeberria, and J. Zander, Automatic generation of test system instances for configurable cyber-physical systems, Software Quality Journal (2016) 1-43.

[137] H. Abbas, B. Hoxha, G. Fainekos, and K. Ueda, Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems, in Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on. pp. 1-6, 2014.

[138] B. K. Aichernig, H. Brandl, E. Jöbstl, and W. Krenn, Model-based mutation testing of hybrid systems, in International Symposium on Formal Methods for Components and Objects. pp. 228-249, 2009.

[139] H. Roehm, J. Oehlerking, M. Woehrle, and M. Althoff, Reachset conformance testing of hybrid automata, in Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control. pp. 277-286, 2016.

[140] T. Dang, and T. Nahhal, Coverage-guided test generation for continuous and hybrid systems, Formal Methods in System Design, vol. 34, no. 2 (2009) 183-213.

[141] T. Dang, and N. Shalev, State estimation and property-guided exploration for hybrid systems testing, in IFIP International Conference on Testing Software and Systems. pp. 152-167, 2012.

[142] Y. Qin, C. Xu, P. Yu, and J. Lu, SIT: Sampling-based interactive testing for self-adaptive apps, Journal of Systems and Software, vol. 120 (2016) 70-88.

[143] A. J. Ramirez, A. C. Jensen, B. H. Cheng, and D. B. Knoester, Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems, in Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. pp. 568-571, 2011.

[144] E. M. Fredericks, A. J. Ramirez, and B. H. Cheng, Validating code-level behavior of dynamic adaptive systems in the face of uncertainty, in International Symposium on Search Based Software Engineering. pp. 81-95, 2013.

[145] E. M. Fredericks, B. DeVries, and B. H. Cheng, Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty, in Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 17-26, 2014.

[146] M. Harman, Making the Case for MORTO: Multi Objective Regression Test Optimization, in Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011, pp. 111-114.

[147] A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria, Test case prioritization of configurable cyber-physical systems with weight-based search algorithms, in Proceedings of the 2016 on Genetic and Evolutionary Computation Conference (GECCO), Search-Based Software Engineering (SBSE) track. pp. 1053-1060, 2016.

[148] A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria, Search-based test case selection of cyber-physical system product lines for simulation-based validation, in Proceedings of the 20th International Systems and Software Product Line Conference (SPLC). pp. 297-306, 2016.

[149] M. Harman, A. Mansouri, and Y. Zhang, Search Based Software Engineering: Trends, Techniques and Applications To Appear in ACM Computing Surveys (2012).

[150] D. Pradhan, S. Wang, S. Ali, T. Yue, and M. Liaaen, CBGA-ES: A Cluster-Based Genetic Algorithm with Elitist Selection for Supporting Multi-objective Test Optimization, in Proceedings of 10th IEEE International Conference on Software Testing, Verification and Validation (ICST 2017), 2017.

[151] "Search-Based Software Engineering Repository," accessed; http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/.

[152] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, TimeAware test suite prioritization, in Proceedings of the 2006 international symposium on Software testing and analysis, Portland, Maine, USA, 2006, pp. 1-12.

[153] L. Zhang, S.-S. Hou, C. Guo, T. Xie, and H. Mei, Time-aware test-case prioritization using integer linear programming, in Proceedings of the eighteenth international symposium on Software testing and analysis, Chicago, IL, USA, 2009, pp. 213-224.

[154] D. Kahneman, and A. Tversky, Prospect theory: An analysis of decision under risk, Econometrica: Journal of the econometric society (1979) 263-291.

[155] OMG, "Structured Assurance Case Metamodel," 2015, http://www.omg.org/spec/SACM/.

[156] ISO/IEC/IEEE, "ISO/IEC/IEEE 24765: Systems and software engineering -- Vocabulary," 2010, http://www.iso.org/iso/catalogue_detail.htm?csnumber=50518.

[157] ISO/IEC/IEEE, "ISO/IEC/IEEE Systems and software engineering -- Architecture description," ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000), 2011, pp. 1-46, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6129467.

[158] ISO/IEC/IEEE, "ISO/IEC/IEEE 15288: Systems and software engineering - System life cycle processes," 2015, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=63711.

[159] ISO/IEC, "ISO/IEC 16085: Systems and software engineering - Life cycle processes - Risk management," 2006.

[160] ISO/IEC, "ISO/IEC 25010: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models," 2011, http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733.

[161] ISO/IEC, "ISO/IEC 15026-x: Systems and software engineering -- Systems and software assurance," 2013.

[162] ISO/IEC, "ISO/IEC 12207: Systems and software engineering - Software life cycle processes," 2008, http://www.iso.org/iso/catalogue_detail?csnumber=43447.

[163] ISO/IEC, "ISO/IEC Guide 98: Uncertainty of measurement," 2009.

[164] ISO/IEC, "ISO/IEC 10165-7: Information technology - Open Systems Interconnection - Structure of management information: General relationship model," 1997, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24160.

[165] IEC, "IEC Guide 115: Application of uncertainty of measurement to conformity assessment activities in the electrotechnical sector," 2007, https://webstore.iec.ch/publication/7524.

[166] IEC, "IEC 31010: Risk management - Risk assessment techniques," 2009, http://www.iso.org/iso/catalogue_detail?csnumber=51073.
[167] IEEE, "IEEE Standard for Software Quality Assurance Processes," IEEE Std 730-2014 (Revision of IEEE Std 730-2002), 2014, pp. 1-138, http://ieeexplore.ieee.org/stampPDF/getPDF.jsp?tp=&arnumber=6835311, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6835311.
[168] IEEE, "IEEE Standard for a Software Quality Metrics Methodology," IEEE Std 1061-1998, 1998, p. i, http://ieeexplore.ieee.org/stampPDF/getPDF.jsp?tp=&arnumber=749159.
[169] IEEE, IEEE P2413: Standard for an Architectural Framework for the Internet of Things (IoT), http://standards.ieee.org/develop/project/2413.html.
[170] ISO, ISO 9000: Quality management systems - Fundamentals and vocabulary, 2015, https://www.iso.org/standard/45481.html.
[171] ISO, "ISO 31000: Risk management," 2009, http://www.iso.org/iso/home/standards/iso31000.htm.
[172] ISO, "ISO 3534-1: Statistics -- Vocabulary and symbols -- Part 1: General statistical terms and terms used in probability," 2006, http://www.iso.org/iso/catalogue_detail.htm?csnumber=40145.
[173] ISO, "ISO 21748: Guidance for the use of repeatability, reproducibility and trueness estimates in measurement uncertainty estimation," 2010, http://www.iso.org/iso/catalogue_detail.htm?csnumber=46373.
[174] ISO/TR, ISO/TR 13587: Three statistical approaches for the assessment and interpretation of measurement uncertainty, (2012).
[175] ISO/TS, "ISO/TS 17503: Statistical methods of uncertainty evaluation - Guidance on evaluation of uncertainty using two-factor crossed designs," 2015, http://www.iso.org/iso/catalogue_detail.htm?csnumber=59900.
[176] ISO, "ISO 9241-x: Ergonomics of human-system interaction," 2010.
[177] JCGM, "JCGM 200: International vocabulary of metrology - Basic and general concepts and associated terms," 2012.
[178] ETSI, "TR 102 422 Methods for Testing and Specification (MTS) IMS Network Integration Testing Infrastructure Testing Methodology (v1.1.1)," 2005, http://www.etsi.org/deliver/etsi_tr/102400_102499/102422/01.01.01_60/tr_102422v010101p.pdf.
[179] ETSI, "ETSI EG 203 120 Methods for Testing and Specification (MTS); Model-Based Testing (MBT); Methodology for standardized test specification development (v1.1.1)," 2013, http://www.etsi.org/deliver/etsi_eg/203100_203199/203130/01.01.01_50/eg_203130v010101m.pdf.
[180] ETSI, "ETSI TR 101 583 Methods for Testing and Specification (MTS); Security Testing; Basic Terminology (v1,1,1)," 2015, http://www.etsi.org/deliver/etsi_tr/101500_101599/101583/01.01.01_60/tr_101583v010101p.pdf.
[181] ETSI, "ES 201 873-x TTCN-3 Specifications," http://www.ttcn-3.org/index.php/downloads/standards.
[182] ETSI, "ETSI ES 203 119-x Methods for Testing and Specification (MTS); The Test Description Language (TDL); ," http://www.etsi.org/technologies-clusters/technologies/test-description-language.
[183] ETSI, "ETSI TR 102 840 Methods for Testing and Specifications (MTS); Model-based testing in standardisation (v1.2.1)," 2011, http://www.etsi.org/deliver/etsi_tr/102800_102899/102840/01.02.01_60/tr_102840v010201p.pdf.
[184] OASIS, "Open Services for Lifecycle Collaboration Core Specifications Version 3.0," 2013, http://open-services.net/wiki/core/Specification-3.0/OSLCCoreSpecificationsV3/OSLCCoreSpecificationsV3/.

Appendix A. List of Standardization Bodies and Standards with Uncertainty

Body	Modeling/Testing/MBT/Other	Standard	Uncertainty
OMG	Modeling	Unified Modeling Language (UML) [54]	No
		UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems [56]	**Yes (Probability)**
		Systems Modeling Language (SysML) [55]	**Yes (Probability)**
		Object Constraint Language (OCL) [57]	No
		MetaObject Facility (MOF) [58]	No

		Structured Assurance Case Metamodel (SACM) [155]	**Yes (Evidence, Confidence, Confidence Level)**
	MBT	UML Testing Profile (UTP) [53]	No
ISO, IEC and IEEE	Modeling	ISO/IEC 10746 series – Reference model of Open Distributed Processing (RM-ODP) [62]	No
		ISO/IEC 19505 series – OMG UML 2.4.1 [59]	No
		ISO/IEC 19507:2012 – OMG OCL 2.3.1 [60]	No
		ISO/IEC 19506:2012 – OMG Architecture-Driven Modernization (ADM) — Knowledge Discovery Meta-Model (KDM) [61]	No
	Testing	ISO/IEC/IEEE 29119 series – Software Testing Standard [65]	No
		ISO/IEC 9646 series – Open Systems Interconnection (OSI) - Conformance testing methodology and framework [66]	No
		IEEE 1012-2012 – System and Software Verification and Validation [67]	No
		IEEE 829-2008 – Software and System Test Documentation [68]	No
		IEEE SA - 1008-1987 – IEEE Standard for Software Unit Testing [69]	No
		IEEE 1044-2009 – Classification for Software Anomalies [70]	No
	Other	ISO/IEC/IEEE 24765:2010 – Systems and software Engineering – Vocabulary [156]	No
		ISO/IEC/IEEE 42010:2011 – Systems and software Engineering - Architecture description [157]	No
		ISO/IEC/IEEE 15288:2015 – Systems and software engineering- System life cycle processes [158]	No
		ISO/IEC 16085:2006 – Systems and software Engineering - Life cycle processes - Risk management [159]	No
		ISO/IEC 25010:2011 – Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models [160]	No
		ISO/IEC 15026 series – Systems and software assurance [161]	No
		ISO/IEC 12207:2008 – Systems and software engineering - Software life cycle processes [162]	No
		ISO/IEC Guide 98 series – Uncertainty of Measurement [163]	**Yes**
		ISO/IEC 10165-7:1996 – Open Systems Interconnection (OSI) - Structure of management information: General relationship model [164]	No
		IEC Guide 115:2007 – Application of uncertainty of measurement to conformity assessment activities in the electrotechnical sector [165]	**Yes**
		IEC 61508:2010 – Functional safety of electrical/electronic/programmable electronic safety-related systems [71]	**Yes (Probability)**
		IEC 31010:2009 – Risk Assessment Techniques [166]	No
		IEEE 730-2014 – Software Quality Assurance Processes [167]	No
		IEEE 1061-1998 – Software Quality Metrics Methodology [168]	No
		IEEE P2413 – Standard for an Architectural Framework for the Internet of Things (IoT) [169]	No
		ISO 9000 series – Quality Management [170]	No
		ISO 31000 – Risk Management [171]	**Yes (Risk, Uncertainty, Effect, Likelihood)**
		ISO 3534-1:2006 – General statistical terms and terms used in probability [172]	**Yes**
		ISO 21748:2010 – Guidance for the use of repeatability, reproducibility and trueness estimates in measurement uncertainty estimation [173]	**Yes**
		ISO/TR 13587:2012 – Three statistical approaches for the assessment and interpretation of measurement uncertainty [174]	**Yes**
		ISO/TS 17503:2015 – Guidance on evaluation of uncertainty using two-factor crossed designs [175]	**Yes**
		ISO 9241 series – Ergonomics of human-system interaction [176]	No
JCGM	Other	JCGM 200:2012 – International vocabulary of metrology- Basic and general concepts and associated terms (VIM) [177]	**Yes**
ETSI	Testing	ETSI TR 102 422 V1.1.1 (2005-04) – IMS Network Integration Testing Infrastructure Testing Methodology [178]	No
		ETSI EG 203 130 V1.1.1 (2013-04) – Methodology for standardized test specification development [179]	No

		ETSI TR 101 583 V1.1.1 (2015-03) – Security Testing; Basic Terminology [180]	No
		ETSI ES 201 873 series on TTCN-3 [181]	No
		ETSI ES 203 119 series on Test Description Language (TDL) [182]	No
	Testing, MBT	ETSI TR 102 840 V1.2.1 (2011-02) – Methods for Testing and Specifications (MTS); Model-based testing in standardization [183]	No
		ETSI ES 202 951 V1.1.1 (2011-07) – Methods for Testing and Specification (MTS); Model-Based Testing (MBT); Requirements for Modeling Notations [63]	No
		ETSI EG 201 015 V2.1.1 (2012-02) – Methods for Testing and Specification (MTS); Standards engineering process; A Handbook of validation methods [64]	No
OASIS	Other	Open Services for Lifecycle Collaboration (OSLC) [184]	No