Journal of Software Engineering and Applications, 2015, 8, 499-519
Published Online September 2015 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2015.89048

Review of Model-Based Testing Approaches in Production Automation and Adjacent Domains—Current Challenges and Research Gaps

Susanne Rösch[1], Sebastian Ulewicz[1], Julien Provost[2], Birgit Vogel-Heuser[1]

[1]Institute of Automation and Information Systems, Technische Universität München, München, Germany
[2]Assistant Professorship for Safe Embedded Systems, Technische Universität München, München, Germany
Email: roesch@ais.mw.tum.de, ulewicz@ais.mw.tum.de, vogel-heuser@ais.mw.tum.de, provost@ses.mw.tum.de

Received 18 August 2015; accepted 27 September 2015; published 30 September 2015

Abstract

As production automation systems have been and are becoming more and more complex, the task of quality assurance is increasingly challenging. Model-based testing is a research field addressing this challenge and many approaches have been suggested for different applications. The goal of this paper is to review these approaches regarding their suitability for the domain of production automation in order to identify current trends and research gaps. The different approaches are classified and clustered according to their main focus which is either testing and test case generation from some form of model automatons, test case generation from models used within the development process of production automation systems, test case generation from fault models or test case selection and regression testing.

Keywords

Model-Based Testing, Automated Production Systems, Conformance Testing, Regression Testing, Fault Injection, Survey

1. Introduction

Testing is the "activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component" [1]. As systems in

How to cite this paper: Rösch, S., Ulewicz, S., Provost, J. and Vogel-Heuser, B. (2015) Review of Model-Based Testing Approaches in Production Automation and Adjacent Domains—Current Challenges and Research Gaps. *Journal of Software Engineering and Applications*, **8**, 499-519. http://dx.doi.org/10.4236/jsea.2015.89048

industrial automation are becoming more complex [2], the challenge of validation has also gained significance. The high standards regarding non-functional requirements such as quality, timing and safety aspects [3] of automated production systems or conformance test of critical controllers as advocated by certification bodies and standards [4] [5] further increase the challenge. In [6], it is shown that in currently established tools for the development of industrial control software, testing activities are rarely automated and have to be conducted manually. Furthermore, "the process of deriving tests tends to be unstructured, not reproducible, not documented, lacking detailed rationales for the test design, and dependent on the ingenuity of single engineers" [7]. Consequently, many works have been conducted in research on the key challenges of improving and automating the testing process in various domains of software engineering including Programmable Logic controller (PLC) software engineering in production automation. Model-based testing is a possible solution to automate the generation of tests.

"Model-based testing (MBT) is a variant of testing that relies on explicit behaviour models that encode the intended behaviours of an SUT and/or the behaviour of its environment. Test cases are generated from one of these models or their combination, and then executed on the SUT." [7]. Methods aiming towards this goal can be part of model-based and model-driven development processes, which are increasingly established [8] [9].

The limiting factor of model-based testing, the same as in code generation, is of course that faulty models, which may be due to false requirements, result in faulty test cases. This emphasizes the importance of research on proving the correctness of models [10], which is not focused on in this paper. In [2], today's challenges of software engineering in automation are presented. In this paper, the challenges are considered in the context of model-based testing.

- The first challenges that are mentioned regard usability considering the different stakeholders in automation. It is stated that software applications are mainly built and implemented by mechanical and electrical engineers using mainly IEC 61131-3. Therefore, the research question 1) in this paper is *the development of approaches that are suited for generating test cases which may be executed against IEC* 61131-3 *applications*.
- The challenge of changeability during runtime and life-cycle management, which includes many software changes during runtime, leads to the research question 2) *of adequate support during regression testing in automation*.
- Research question 3) is derived from the challenge that "additional system aspects have to be considered in the design of automation software, e.g. the causal and temporal associations, resulting from the automation-affected technical process and physical structure of the technical system" [2]. Therefore, *testing approaches must take testing of faults and failures that may occur from the technical process and technical system, such as hardware effects, into account and provide methods of testing them.*
- Another important aspect that must be considered in automation are 4) *real-time constraints which must be handled by the automation platforms.* When testing, hardware-in-the-loop testing approaches address this topic, as they are executed on the actual automation architecture.
- Last but not least, the challenge of 5) *adequate models including the different disciplines is mentioned.* When regarding the field of model-based testing exactly the same research question applies.

The complementary techniques to testing, which are based on the validation that the implementation behaves equivalently to its specification, are formal verification, model-checking and theorem-proving, which prove "that the internal semantics of a model is consistent, independently from the modeled system" [11]. However, as stated in [10], verification is rarely established within the industry in contrast to testing in computer science [10], which is also true for production automation. Consequently, this field of research is not in the focus of the survey, but a comprehensive overview on formalization of models for verification purposes may be found in [12] and [13] for the interested reader.

The goal of this paper is to review current model-based testing approaches in production automation and close adjacent domains. On the one hand, adjacent domains related to disciplines are considered, which are also part of production automation, namely software engineering and computer science, electrical/electronical and mechanical engineering. On the other hand, domains also including these three disciplines, such as embedded systems were considered. Adjacent domains were looked into, in order to identify promising approaches which might be adopted, put current trends into context and define current challenges and research gaps. This paper focuses on validation of functional and non-functional requirements concerning the PLC application software rather than structural analysis, stress testing or performance analysis [14]. Also, the systems' software behavior is regarded as deterministic, excluding stochastic testing.

The remainder of this paper is structured as follows. In the following section, the research methodology on how this survey was conducted is explained. Further on, classification criteria are established in order to distinguish and structure current research approaches. In the subsequent sections, the different research fields are discussed, structured based on the classification criteria concerning the testing goal and the models used as a basis for test case generation. In Section 4, testing of discrete event system models and test generation from discrete event system models, targeting the research Questions 1 and 4, are described. Then, test case generation from models used in the development processes, including approaches which target research Questions 1, 4 and 5, is explained in Section 5. Test generation based on technical system and technical process fault models (research Questions 1 and 3-5) is explained in Section 6. Change impact analysis of specification or code models for regression testing (Section 7) (research Question 2) completes the discussion of the different research fields. Subsequently, research gaps in model-based testing in production automation are identified and a conclusion is given.

2. Research Methodology, Design and Classification

The primary goal of this work was to find approaches, where model-based testing approaches have been introduced in the production automation domain, respectively testing of PLC control software, and the research questions introduced in the introduction have been addressed. This has especially been done for the topics of generation of tests from models used in automation, particularly taking models specified in [15], such as the UML, into account. However, if little work could be found on the specific research questions in the domain of production automation, as was the case for test generation from models focusing on faults from the technical process (research Question 3) and regression testing (research Question 2), adjacent domains have been investigated in order to find promising approaches that may be adapted and applied to the production automation domain. The main databases which have been searched include “IEEE Xplore/Electronic Library Online (IEL)”, “ScienceDirect” and “SpringerLink”. The main search terms included “model-based testing”, “conformance testing”, “regression testing” and “fault injection” in combination with domain specific terms such as “production automation”, “PLC”, “factory automation”, “manufacturing automation” and “machine and plant automation”. The papers have then been classified according to the methodology described in Section 3. The survey is not exhaustive, having not classified every paper that was found by the search described above, but rather aims at giving an overview on the current state of the art and open research issues.

3. Methodology, Classification and Definitions

Recently model-based testing and related terms and definitions such as test case, test selection criteria and test case specification have been updated and defined in [7]. In general model-based testing uses explicit behavior models that specify the intended behavior against an SUT. Test cases are generated from one of these models or their combination, and then executed on the SUT. The classification criteria for model-based testing approaches proposed in this paper use these and other definitions from [16] and [1] related to testing and put them into context regarding the key research questions in the field of production automation. An overview of the classification criteria (*C1-C5)* may be found in **Figure 1**.

3.1. Model/Specification—Classification Criterion C1

To validate automation systems, different measures are taken regarding the *testing goal* which differs according to the type and information included in the *specification* (C1b) also called *model paradigm* (*C*1*a*). If only the expected *behavior* (*C*1*a-i*) is specified, the *functional compliance* may serve as the testing goal. If fault models do exist, the reliability of the system, which is determined by the reaction to faults, can be tested and research Question 3 is targeted. This can be done for example by using fault injection [17] (see Section 6). Another research field is concerned with defining and analyzing change models and therefore focuses on regression testing tackling research Question 2 (see Section 7).

Furthermore, the different works can be distinguished by the kind of *specification* (*C*1*b*) model that is used. Many approaches require discrete event system specifications to generate test cases (see Section 4) while others focus on providing more user friendly modeling languages for system engineers in order to specify the system (research Question 5, see Section 5). The latter often include a process to formalize the modeling languages further to be able to generate test cases.

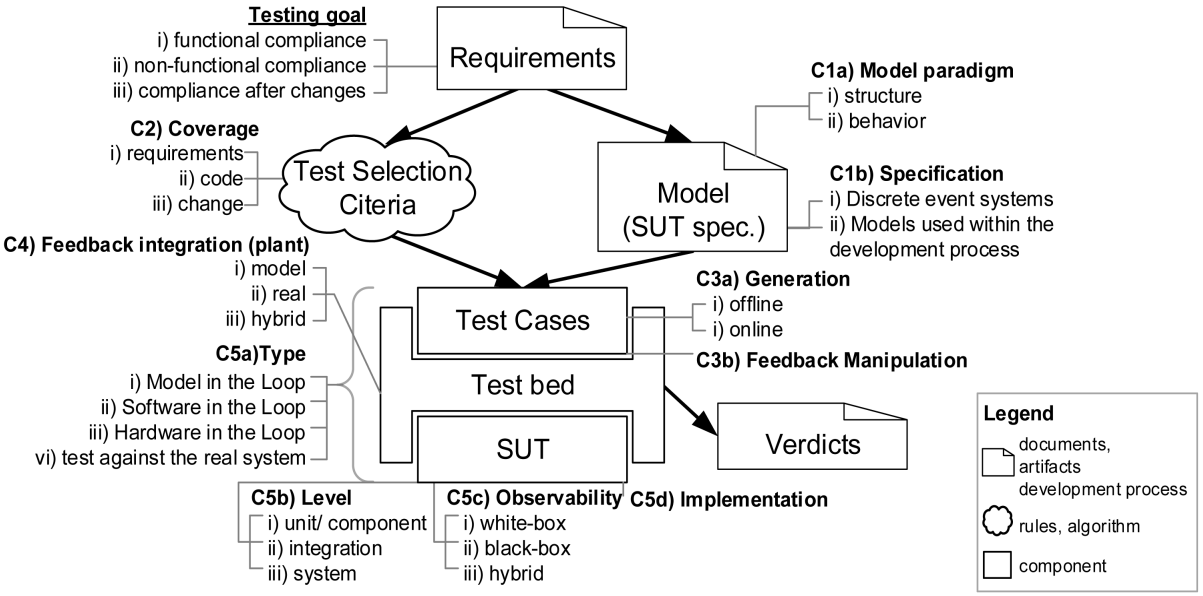

Figure 1. Overview model-based testing and classification based on [7].

3.2. Test Selection Criteria—C2

The test cases are generated based on the specification, *i.e.* models, and on *test selection criteria* which "define [...] the facilities that are used to control the generation of tests" [7]. As an example, when a state machine is used as model/specification (C1a-i, C1b-i), several test cases may be created using a *coverage criterion* (*C2a*) such as transition coverage. For black-box testing, a typical criterion would be data coverage. Further test selection criteria may be found in [7].

3.3. Test Cases—C3

Besides test case generation based on a predefined model and static test selection criteria, sometimes test cases are generated "on-the-fly" by comparing the outputs emitted by the *system under test* (SUT) during a test run to the expected ones according to the specification and adapting the test cases depending on the local *test verdict* (pass or fail). This type of test generation and execution is called *online test generation* (*C3a-ii*). On the opposite side, *offline test generation* (*C3a-i*) refers to the separation of the generation and execution. If the test case also manipulates the input from other test components, such as the injection of faults, this is called *feedback manipulation* (*C3b*).

3.4. Test Bed—C4

The execution of test cases against the SUT is made possible by the *test bed* which is the "environment containing the hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test" [1] (research Questions 1 and 4). One determining factor of the test bed is the kind of environment feedback it includes. Common test cases usually simply consist of predefined inputs and the SUT's outputs which are compared to the expected ones. The input and expected output signals are static, *i.e.* they are not influenced by the SUT's behavior, and are specified using implicit knowledge or assumptions about the SUT's environment's behavior. In contrast to this, the SUT's output signals can be dynamically fed back into its inputs through a function representing the SUT's environment's behavior, e.g. a plant's behavior represented by a *model* (*C4-i*), *i.e.* simulation. On the one hand, test sequences for including complex environment behavior are simplified, shortened and can be based on physical correlations, rather than implicit knowledge. On the other hand, extensive effort has to be invested into the model's definition and verification.

Other test beds include a setup with a connection to the *real* (*C4-ii*) plant (hardware), or some parts of it, in order to include the actual feedback in the test execution.

3.5. System under Test (SUT)—C5

The setup of the SUT with the test bed determines which *type* (*C*5*a*) of test is conducted.

Some approaches do only test models of the implementation (*Model in the Loop—MiL* (*C*5*a-i*)) while other tests are done using the implemented software (*Software in the Loop—SiL* (*C*5*a-ii*)) using static sequences or a simulation.

In [18], two major forms of simulation in industrial automation are identified: System simulation and *Hardware in the Loop—HiL* (*C*5*a-iii*) simulation. In system simulation, both the control software, *i.e.* the SUT, and simulation are running on the same system—usually a standard computer. In this case, the control software is running on a Soft-PLC, which can either implement the simulation or can be connected to an external simulation (SiL). In [19], this type of simulation and its verification are investigated. The second major form is HiL simulation/testing. A "typical HiL setup would include a controller with loaded control code connected to a testing environment" [20]. For automated production systems this means that the control software is executed on the target hardware, *i.e.* a real PLC, and connected to the simulation system via a field bus. The field of HiL has become broader as in "past years [...] HiL has expanded to encompass component testing as well" [20].

The automatic execution of test cases is also a field where little work may be found in the context of production automation. As mentioned in [6], full support of current tools for PLC platforms is still missing. Approaches that integrate the automatic execution in available tools or new tools targeted at PLC platforms are scarcely available.

While MiL approaches do help to root out faults in the early design phases and SiL approaches root out faults during the implementation phase [17], only HiL approaches enable the validation of the integrated system and the inclusion of hardware related effects (research Question 3), as every model is some form of abstraction and only the system itself is completely accurate.

The SUT also determines the testing *level (C5b)*, which ranges between *unit/module/component test* (*C*5*b-i*), *integration testing* (*C*5*b-ii*) and *system testing* (*C*5*b-iii*). Testing approaches are needed for all stages of the development process.

Furthermore, the observability of the implementation must be considered (C5c): White-Box testing is "a type of testing in which you examine the internal structure of a program" [21] and therefore the implemented code is used as a basis for test case generation. Black-Box testing is an approach where the "internal structure is ignored. Test data are derived solely from the application's specification" [21].

In this paper, approaches are furthermore distinguished by their applicability in the field of production automation. Some approaches are developed for testing *implementations* (*C*5*d*) of languages which are designed for use in production automation systems such as the IEC 61131-3, which is established within industry right now [22], or the IEC 61499 which has been advocated by many researchers [23]. Other approaches have been developed in different domains and therefore for different languages but may be applicable in the field of production automation.

The different validation techniques aim at detecting faults in an SUT before the system is commissioned for operation. *Fault*, *error*, and *failure* are defined in [24]. A *failure* refers to "an event that occurs when the delivered service deviates from correct service", which was originally specified or expected. The *failure* is caused by a deviation from the expected system state. This "deviation is called an error" and "the adjudged or hypothesized cause of an error is called a fault".

3.6. Test Verdict

After the execution of a test sequence (once a final state has been reached), the test verdict is reported and documented which can either be:

- *pass*: the equivalence relation between the specification and the implementation model is fulfilled: the implementation conforms to its specification
- *fail*: the equivalence relation is not fulfilled and counter-examples (or traces) can be given: the implementation does not conform to its specification
- *inconclusive*: the execution on the test has not permitted to assess the equivalence of the two models neither to give counter-examples. This case corresponds to test execution where the observed behavior of the implementation does not lead to a *fail* state (no counter example can be found), but where a part of the behavior to be tested could not be observed. For more details about this verdict in the context of on-the-fly test case

generation see [25].

In order to avoid false positive and false negative verdicts, a conformance test must be:

- *valid* (or *exhaustive*): "the test suite comprises all combinations of input values and preconditions" [16], therefore every implementation which does not conform to its specification must be detected and rejected;
- *non-biased* (or *sound*): no implementation which conforms to its specification should be rejected.

4. Testing and Test Sequence Generation of and from Discrete Event System Specifications

4.1. Testing of Discrete Event System Models

A promising solution to develop testing techniques is to benefit from the results of the researches of the Computer Science and Discrete Event Systems communities in the domain of conformance test of formal models. In these works, the specification is given in the form of a formal model such as a finite state machine [26], a transition system [27]-[29], a timed automaton [30] or a Petri net [31]. The implementation is supposed to behave according to the same formalism, e.g. if the specification is defined by a Moore machine, the implementation should also behave like a Moore machine and not like, for instance, a Mealy machine. The interested readers are referred to the above-mentioned paper to get more details about these formal languages. The goal of these testing techniques is then to validate the inclusion or equivalence relations between the two models. The equivalence between a specification model and an implementation model are usually tested according to their *observable behavior* and their *trace equivalence*. Below are some examples of conformance relations between an implementation *i* and its specification *s*, for formal definitions and details about these relations the interested readers are referred to [27] [28]:

- $i \leq_{tr} s$: *i* conforms to *s* wrt. the relation $\leq_{tr}$ if and only if for all test sequences composed of the input alphabet of the models, the traces of observable actions of *i* are included in the traces of observable actions of *s*. This relation does not consider inputs and outputs but observable actions.
- *i* **ioconf** *s*: *i* conforms to *s* wrt. the relation **ioconf** if and only if for all test sequences *generated from the traces of s*, the set of observable outputs of *i* is included in the set of observable outputs of *s*. This relation allows partial specifications because the test sequences are generated only from the traces of the (partial) specifications.
- *i* **ioco** *s*: *i* conforms to *s* wrt. The relation **ioco** if and only if for all test sequences generated from the suspension traces of *s* (traces that also represent the absence of emitted output: specified quiescence, or missing output actions), the sequence of emitted outputs of *i* is included in the sequence of emitted outputs of *s*.

In the context of model-based testing of automated production systems, the use of conformance relations defined over the observable input/output relations is more appropriate because the internal behavior of the controller cannot always be observed (black-box testing). Recently, [32] proposed a new conformance relation for model-based testing of PLCs.

As mentioned earlier a test sequence can either be generated offline or online. In the first case, the test sequence is a straight sequence of input/output couples. In the second case, the continuation of the test execution depends on the observed outputs from the previous test step. Both cases can be modeled as state-machine or labeled transition systems where each final state defines the test verdict.

Since the verdict of a test is based on the observation of the behaviors of two models (specification and implementation), an important issue is to be able to ensure the state synchronization and the state identification of those models. A review of the usual state identification and synchronization techniques is presented in [26] [33]-[35]. Even though the basic techniques are well established, several research activities are still conducted on the improvement of those techniques [36] [37] and their application to others formal models [38].

Conformance of an implementation to its specification requires that a *test objective* or *test selection criteria* be first defined. A classical test objective, when critical systems are considered, is to cross at least once each edge of the directed graph that represents the structure of the formal model; this permits to check every state change from each state of the formal model. Then, the test sequence can be constructed from this model. However, depending on the scale of the system, the security level and the assumptions made on the implementation, different test objectives can be defined. **Table 1** lists different techniques that can be applied depending on the assumptions made on the implementation. A more complete overview of the main testing challenges and the different testing techniques developed to improve the test coverage and the reliability of the test results is given

Table 1. Assumptions on formal models.

Assumptions	Proposed solutions	References
The implementation can have extra states	Bounded number of extra states	[40]
Time	Test boundaries (maximum duration of a timer)	[41]
Variables within a range	Domain testing and/or boundary testing	[42]
Variables within a range	Symbolic approaches	[43]-[45]
Partial specifications	Verifying partial inclusion: $\text{Impl.} \subseteq \text{Spec1} \wedge \text{Impl.} \subseteq \text{Spec2} \wedge \cdots$	[46]
Partial specifications	Inferring partial models	[47]

in [39]. It is also of importance to note that most of these works have been developed using event-based formalism (vs. signal-based formalism).

As testing is based on the validation of the equivalence (or inclusion relation) of two behaviors (an implementation and its specification), testing can be seen as an exploration process, which permits to explore the behavior of an implementation and compare it to its specification, and requires the execution of the implementation. In contrast to testing, model-checking is based on the verification of properties that should hold for a behavior. Model checking can be seen as a confirmation process, it is used to confirm that a property holds or not for the whole behavior. Symbolic approaches are used to handle the scalability of verification techniques. Even though symbolic approaches cannot be applied during the execution phase of testing (during the execution, for each test step, the implementation is solicited with a set of fixed values, not with a set of value ranges), model-checking techniques can be used during the first phase to generate test sequences [48]-[52].

4.2. Model Transformation

In order to apply the fruitful theoretical results on specification and implementation used in the automation industry there is a need to transform industrial models into more formal models. Several works have considered the issue of model transformation from industrial (or standardized) models into (semi-)formal models in order to apply existing formal techniques and tools. **Table 2** gives an overview of existing works on such model transformations.

Many of the transformations presented in **Table 2** have been developed in the context of verification. Some of the existing methods only consider the source code of the input language while some others also consider a model of the PLC operation. When only the source code is considered (*i.e.* without a model of the controller execution), the input model can be considered as an infinitely reactive model (*i.e.* the response time delay equals zero). Thus, the transformation to another formal model can be facilitated by the use of their meta-model. For instance, model-to-model transformations can be obtained using the model transformation tool ATL [78].

The transformation into formal models also permits the combination of the model of the software, with a model of the execution of the controller—if not already done—and also with a model of the plant that is to be controlled. The more information the composed model contains, the more reliable the simulation and the results of the verification and validation methods is [79].

5. Test Sequence Generation from Models Used within Development Processes

Specification notations such as timed automatons, etc. are up to now rarely used within industry. Mostly partial models or specifications, and informal requirements specifications are used. To bridge this gap and to support system and test engineers in creating models for testing, modeling languages and notations used within the development process are further developed and formalized to receive a basis for test case generation (see **Table 3**). Furthermore, model-based approaches are increasingly applied and developed in production automation. Using similar models for test case generation is the logical next step to further support and improve the development process.

The Unified Modeling Language (UML) was the language designed for usage in the development process [80], where most approaches (and papers) in the context of MBT in production automation could be found. In [81] and [82], useful diagrams for modeling and deriving test cases from the UML are identified for the field of

Table 2. Model transformation from industrial or standardized languages into formal models.

Input language	Output formal model	Source	Remarks
IL (IEC 61131-3), ST (IEC 61131-3)	Timed Net Condition/ Event System, Petri Net, Timed automaton, SIGNAL equations, Model-checker language (SMV)	[53]-[58]	Most of the transformations are performed for verification purposes. The selected references consider the cyclic behavior of industrial controllers such as PLC for the execution of IL programs.
LD (IEC 61131-3)	Time Petri Net, Model-checker language (UPPAAL automata)	[59]-[61]	Most of the transformations are performed for verification purposes. The selected references consider the cyclic behavior of industrial controllers such as PLC. [61] even considers multitask systems.
FBD (IEC 61131-3), CFC, IEC 61499	SIGNAL equations, Esterel, Interface automata, Model-checker language (UPPAAL automata, SMV)	[50] [57] [62]-[66]	Most of the transformations are performed for verification purposes. The selected references consider the cyclic behavior of industrial controllers such as PLC. [66] presents a modeling of each block by an interface automaton, the focus is placed on the IO relations: this approach could be adapted to testing of sub-components where only the IOs can be observed (black-box testing).
SFC (IEC 61131-3)	Timed automata, Model-checker language (SMV)	[67]-[70]	SFC (61131-3) is a graphical language with hierarchical relations used to represent mainly sequential behaviors. The semantics defined in the standard contain ambiguity. An improved semantics is proposed in [70].
GRAFCET (IEC 60848)	Monolithic automaton, Mealy machine, Petri Net	[71]-[75]	Grafcet (60848) and SFC (61131-3) share similarities: SFC has been defined from Grafcet. The main difficulty with Grafcet is the stability research. The method presented in [71]-[73] is dedicated to black-box testing and stresses on logic input/output relations.
Matlab Stateflow	Structural operational semantics, Continuous-Time Markov Chains	[76] [77]	The main issue with Matlab State flow models is the interpretation of junction symbols. Depending on the guards before and after a junction symbol, some transitions can be partially fired and their associated actions be executed; if no destination state can be reached after a junction, previous transitions are then backtracked to return to the previous active state without undoing the associated actions.

Table 3. Test generation from models used within the development process.

Source	Domain	C1) Specification	C2) Test Selection Criteria	C3) Execution and Feedback Manipulation	C4) Feedback integration	C5) Test type, Implementation, Observability and Level
[81]-[83]	Production automation	UML (structure: system and context; interaction, mainly state charts: generation of specific test cases)	Depending on diagram, extraction of sequences from interaction diagrams (transition parameter variation, path coverage)	Offline	As spec.	IEC 61499
[84]	Production automation	UML state charts	Model transformation, path unfolding, path coverage	Offline	As spec.	n.a.
[85]	Production automation	UML sequence diagram	Only the specified scenario	Offline	Both SiL and HiL possible	IEC 61131-3, unit

automation software development and especially for IEC 61499 implementations. Structure diagrams such as component diagrams are used to model the context and the interfaces of the SUT. Interaction diagrams are recommended for the extraction of test sequences. In [82], the extraction of test sequences from state charts using round-trip path coverage is shown. A first application of the recommended test case generation process using state chart diagrams especially for IEC 61499 applications is shown in [83].

In [84], an approach to automatically generate test cases from the UML state charts by first transforming them into a formal model (extended safe place/transition nets) is introduced. In order to make the transformation possible, some restrictions on the used model elements are done. Given the formal model, the test case generation is easily made possible using methods such as unfolding the nets.

Making UML models, and in this work especially sequence diagrams, executable is another focus of using UML diagrams in the testing process. In [85], the semantics of sequence diagrams are adapted in order to make direct IEC 61131-3 code generation possible. In this way the modeled test scenarios can be executed directly.

As UML models are already a wide-spread notation also for testing, organizations have started to standardize the language in the context of testing using the profiling mechanism of the UML. The UML Testing Profile (U2TP) has standardized the way to specify the SUT, its context and the specific test cases. The test case scenarios are modeled using the UML sequence diagrams. To make these test cases executable, a transformation from the U2TP to the Testing and Test Control Notation (TTCN-3) has been proposed by [86], which has been established especially in the field of communication. However, up to now, no approaches could be found that have evaluated the applicability of the U2TP in the field of production automation software. In [87], UML test case generation approaches from state charts are combined with the aim of making them executable by mapping them to the TTCN-3. The evaluation of the approach is done using a simple communication protocol but the extension of the approach in order to test PLC control software applications is planned as well.

In recent years the Systems Modeling Language (SysML) is increasingly established for supporting the development process of real-time systems [88].

However, investigations on the possibilities to derive test cases from these models or adapting these models are still missing. Another interesting development that the testing community could benefit from is the improvement of the communication between tools. MATLAB/Simulink models is also increasingly applied in the production automation domain (e.g. [89]). Up to now, the models are mainly used for simulation and code generation [90]. In the automotive domain MBT approaches have been introduced [91] [92]. The applicability in production automation needs to be analyzed.

In [93], an approach to automatically consolidate different domain models from the field of production automation to receive a correct model using AutomationML and MathML is presented. The generation of test cases from such models is still an open topic though.

6. Test Generation from Fault Models—Testing of Unintended Behavior

An important topic that must be addressed when testing and validating automated production systems besides the intended behavior is the reaction to faults that may occur without the system, *i.e.* stemming from the technical system and the technical process, as this also determines the reliability. The faults that must be regarded are causes of failures of automation systems such as the failure of hardware or influences of the environment. These faults must be handled by the software of automation systems by error handling routines. To prove the validity and correctness of systems, fault injection (FI) is a method that has been established in order to measure the dependability. Fault injection is used as a means to evaluate error handling mechanisms concerning fault detection and error handling. FI approaches can be divided into hardware-implemented FI (HWIFI), where faults are for example injected by forcing pins, software-implemented FI (SWIFI), where faults of the system are emulated by the software, and model-implemented FI (MIFI) also called simulation-based FI [94]. While HWIFI and SWIFI are mostly used on prototypes or for system testing, MIFI is rather used in earlier conceptual and design phases to give early feedback to engineers [17]. It is possible that the approach implements one kind of fault injection (e.g. MIFI) but still is another kind of test (e.g. SiL), because the fault might be injected by the test case through the test bed (e.g. fault is injected in the simulation model but the SUT is the implemented software not running on the final system). FI is determined by the *faults* (*F*) that are injected, a set of *activations* (*A*), defining the triggers for the injection, the *readouts* (*R*), *i.e.* the logging of the system reaction or the outputs, and the actions or measures (M) that are derived from the analysis of F, A and R, as defined in [95].

A *fault model* defines the types of possible faults of a system in respect to several different criteria such as the phase of creation (design, implementation, etc.), the dimension (hardware faults, software faults), the system boundary (introduced from within or without the system, etc.) [24]. The type of faults *F* commonly injected by FI, are mostly hardware faults in contrast to software faults, human made faults, etc. (for classification of faults see [24]). Another aspect considered by many approaches is the timing behavior of the fault which is either permanent, which means that it occurs and is permanent from this point on, or if it occurs only at certain time intervals or at random. The activation *A* is differentiated whether the fault is injected before runtime or during operation. By injecting faults during operation the opportunity to activate the fault by time-triggered or event-triggered conditions is created, making it possible to realize more complex fault scenarios. The three different approaches—HWIFI, MIFI and SWIFI are further explained in the following subsections. An overview is shown in **Table 4**.

6.1. Hardware-Implemented Fault Injection (HWIFI)

Many tools and methods for testing the reaction to faults from integrated circuits and especially microprocessors have been established. The methods vary between FI with contact such as the injection of faults on pin level and FI without contact such as heavy-ion-radiation or electromagnetic interference [97]. A third means of introducing faults is the use of built-in logic especially designed for the SUT [96]. The specification is based on faults *F* which typically occur within (micro-) processors such as bridging faults, stuck-at faults, bit flips or power surges [17]. In the field of electrical engineering the fault models are still being updated and further developed [105]. For methods that are controllable and reproducible such as pin-level fault injection the activation *A* is possible to be time-triggered or event-triggered. An overview and more detailed descriptions of the different tools and methods may be found in [17] and [97].

Hardware-in-the-loop test benches have been developed for more complex systems such as PLCs, where the PLC is for example connected with a simulation environment [106]. However, no special attention has been given on FI techniques in this field and studies illustrating which faults and fault models would be useful to inject in such systems and the benefits that might be gained from such testing techniques are not available.

6.2. Model-Implemented Fault Injection (MIFI)

In respect to MIFI a distinction can be made whether faults are injected into hardware models or into software models. Hardware models in the domain of electrical engineering are usually modeled using the Very High Speed Integrated Circuit Hardware Description Language (VHDL) and therefore targeted at integrated circuits. The faults that are to be injected are also based on faults which may occur within microprocessors as mentioned in the previous section. These faults are specified and integrated into the model or at the interfaces of the model to simulate faults [98]. In the automotive domain several MIFI approaches have been suggested [94]. These approaches make use of the fact, that MATLAB/Simulink models are commonly used for modeling automotive systems. In [94] it is not only aimed at testing only on simulation level. The result of the test runs against simulation are used for test case generation for the real systems. In the field of production automation an executable UML state chart simulation model is used for FI in [102]. As the approach focuses on testing the application, the program is sliced to extract all possible execution paths leading to a defective component in order to reach full path coverage. The MIFI approaches mostly only have an offline activation *A*, as the faults *F* are predefined within the model.

6.3. Software-Implemented Fault Injection (SWIFI)

As for MIFI and HWIFI, tools for injecting faults in integrated circuits have been evaluated and are available for use [107]. As interfaces and additional functions are standardized for testing and have been introduced in this field, it is additionally made easier to access different locations to inject faults [108]. In [101], a FI approach for embedded system's is introduced in order to validate specified safety functions. The approach targets specific functions which must hold during all circumstances. Specific fault scenarios or the definition of user-friendly notations are not the main focus of the approach. Next to FI during co-simulation, [99] also proposes to use model-based approaches in the automotive domain to introduce faults into the code during code-generation out of Matlab/Simulink models. The components that will be tested as failing are selected in the model, then code is

Table 4. Fault injection in automated production systems and embedded systems.

Source	Domain	C1) Specification	C2) Test Selection Criteria	C3) Execution and Feedback Manipulation	C4) Feedback Integration	C5) Type, Implementation, Observability and Level
[17] [96] [97]	Integrated circuits	Stuck-at, open, complex logical faults, ...	As spec. contact or contactless, time triggered	Offline	HWIFI (pin-level or insertion, heavy-ion radiation, electromagnetic)	Real system, microprocessors/ integrated circuits, system
[98]	Integrated circuits	Fault model (saboteurs, mutants), some mutants generated automatically	As spec.	Offline	MIFI	VHDHL model, integration/system
[94]	Automotive	Fault models, failure mode function (mutants and saboteurs), requirements (Simulink assertion blocks), time window	As spec. minimal cut sets	Offline	MIFI	Matlab/Simulink, unit
[99] [100]	Automotive	Selection of component/ fault nodes in simulation model	As spec.	Offline	MIFI/SWIFI (injection in generated code)	C (generated out of SCADE), unit
[101]	Embedded systems	Mathematical description of functions and safety properties	Mutation operators (insertion of an additional request, reordering of a pair of requests)	Offline	SWIFI	MiL
[102]	Production automation	Fault operators	Path coverage, possible inputs, code splicing, possible paths leading tocomponent	Offline,simulation: UML SC for PLC	MIFI	SiL, IEC 61131-3, unit
[103]	Production automation	Timing sequence diagram	Fault operators (missing signal)		SWIFI	real system, IEC 61131-3, system
[104]	Production automation	Target variable chosen manually, mutant created accordingly, every path checked with every mutant	5 mutation operators according to target		SWIFI	C, FBD, integration

generated where this fault will occur during execution. In [100], a similar approach is suggested using SCADE models. In [104], a method to inject faults during runtime is presented suggesting kernel-based FI on different architectural levels of embedded systems. It is analyzed how and where different faults may be injected in order to test the integration between application, operating system and hardware. The integration of application and operating system is tested by manipulating the communication protocols in between them. The integration testing between operating system and hardware is done using three further mutation operands. If possible the global

variables are manipulated such as communication device errors. If the addresses of the hardware are read-only variables, different mutants (changes to the SUT) are proposed such as disconnecting the hardware or changing the voltage supply for I/O device errors and power-supply. The method seems a viable way to test the integration of the components within embedded systems such as a PLC as proposed in this paper. The test of the control software in respect to hardware faults is not focused on in the paper. [103] suggests a SWIFI approach, where test cases are generated from timing sequence diagrams for the field of production automation. In a preceding survey [103] evaluates timing sequence diagrams as a notation which is commonly used in the domain of production automation. It is assumed that the diagram depicts the expected behavior and any kind of deviation should be handled by the software. Accordingly, test cases with a deviation from the timing sequence diagram (fault operator) are generated. The test execution is done using a setup with the real automation system while injecting the fault directly into the software. The approach is shown to work for processes with discrete behavior. The approach focuses on deviations from the process behavior (*F*: process faults) as sensor values are used for analyzing the behavior. The activation *A* is done during the execution of the system. Communication faults or human made faults are not especially in the focus of this work.

7. Test Selection from Change Models—Regression Testing

During development and operation of automated production systems, the system's software has to be changed and adapted regularly. The changes are categorized in [109] as adaptive, corrective and perfective. Adaptive changes are due to changing environments or requirements, such as changes in hardware or new needed functions. Corrective changes are introduced whenever faults within the software are discovered and fixed. Perfective changes are made during optimization processes, e.g. to shorten production cycle times. Whenever the software is changed, an investigation whether faults are introduced into the software and its compliance to the specification has to be conducted. This process is known as *regression testing*.

The main challenge in regression testing in production automation is to test a changed software system thoroughly, while minimizing the effort to do so. However, if done manually, regression testing is tedious and prone to be incomplete, as dependencies within programs can be intricate. Scenarios leading to errors might be missed and testing efforts are high and have to be repeated with every change. Nevertheless, this type of testing is still dominant in this engineering domain. Model-based testing methods, as described in the previous sections, can help in this regard, by offering ways to automate the test generation. Based on a set of available test cases, regression testing can be conducted by selecting suitable test cases for retesting [110] depending on identified changes, which is done within a *Software Change Impact Analysis* (CIA). The goal of this analysis is to select the test cases that are most likely to find new errors introduced by the changes, but keeping the time of retesting lower than a simple "retest-all" approach, where all tests are re-executed [111]. Therefore, the time needed for the analysis plus the execution of the selected test cases is supposed to be lower than executing all test cases. This selection is done under the assumption that program execution is deterministic and nothing but the code or the specification changes.

CIA can be based on dependencies between software entities, such as functions, classes or statements or on traceable dependencies between the software and other software related artifacts, such as function specifications or interlocking definitions. The former is known as *dependency based change impact analysis*, while the latter is called *traceability based change impact analysis* [112]. An overview of CIA approaches is shown in **Table 5**.

7.1. Traceability Based CIA

This type of CIA is based on the program's specification rather than the changed code itself. The term traceability refers to the ability to trace changes from the specification to its corresponding code through appropriate definitions within the specification. There are different approaches for identifying changes in available specifications and deriving possible influences on test cases for regression testing. As structural models are common in computer science as an artifact for program design, many works take these models to gather information for regression testing regarding integration tests. For selection of integration tests, information about object interdependencies and changes are gathered, which is then used to identify the parts of the program affected by the changes. These entities are then scheduled for regression tests.

A suitable model for this process is the UML Class Diagram, which includes information about the program structure, interfaces and interdependencies between objects. Several works use this model for selecting regression

Table 5. Methods for software change impact analysis.

Source	Domain	C1) Specification	C2) Test Selection Criteria	C3) Generation	C5) Test type, Implementation, Observability
[114] [115]	Computer science	Structure, semiformal (uml class diagram)	Change of specification	None	Integration tests, object-oriented, allows black box
[116]	Automotive, embedded systems	Structure and behavior, component dependency model, uml sequence diagram and directed graph	Change of specification	None	Integration tests, allows black box
[117]	Computer science	Structure and behavior, specification description language	Change of specification	None	Integration tests, object-oriented, allows black box
[113]	Computer science	Structure and behavior, uml sequence diagram, class diagram	Change of specification	None	Integration tests, allows black box
[118]	Computer science	Behavior, extended finite state machine	Change of specification	None	Integration tests
[119]	Computer science	Behavior, call graph	Change of code	None	Integration test, system test, object-oriented, white box
[120]	Computer science	Behavior, program dependency graph	Change of code	None	Integration test, procedural, white box
[121]	Computer science	Behavior, dynamic call graph	Change of code	Previous execution	Integration test, system test, object-oriented, white box
[122]	Computer science	Behavior, dynamic program slice	Change of code	Previous execution	Integration test, system test, object-oriented, white box
[123]	Computer science	Behavior, dynamic control ow graph	Change of code	Previous execution	Unit test, system test, procedural, white box
[124]	Computer science	Behavior, dynamic data ow graph	Change of code	Previous execution	Unit test, integration test, system test, procedural or object-oriented, white box

integration tests [113]-[115]. Other works do not rely on this notation, but create their own formalized model from it, such as the Component Dependency Model [116], or directly specify needed dependencies within a formal description language, such as the Specification Description Language (SDL) [117]. These models focus on relevant dependencies and allow automatic analysis, e.g. graph and dependency analyses, to find relevant changes and assess their impact on testing. In most of the mentioned approaches, certain change classes are defined, which are linked to an influence on test cases. This information is then used to select the test cases.

For regression testing of single units, other specification models are needed, as no information about behavior is stored within class diagrams or other structural models. Several works identify suitable models for defining behavior, such as the UML Sequence Diagram [116], [113], Extended Finite State Machines (EFSM) [118] or the SDL [117]. Again, identification of influences of changes is conducted using change classes, which define whether a test case is influenced or not. Changed entities are identified by comparing the sets of sub-elements of the respective models.

Traceability based CIA is especially interesting, if certain parts of the program code are not accessible (“black box”), e.g. in compiled libraries, as all approaches in the dependency based CIA create a model directly from the

code. This is a common occurrence in production automation, increasing the applicability of the approaches. Yet, the uncommon use of formalized models for software design in production automation hinders an application in this domain. While advantages in later phases can be seen, this domain is still hesitant introducing modeling on a level that can be used for the presented approaches (close enough to the code).

7.2. Dependency Based CIA

This type of CIA can be used for regression testing without relying on models describing structure or behavior of the software. The models needed for analyzing affected entities are directly generated from the code. This can be either done statically, meaning before test execution or dynamically, based on information about previous test execution[1]. Common static dependency analysis methods include building call graphs [119], analyzing which entities call other entities, or program dependence graphs [120], adding information about data dependencies. The resulting graphs include all possible object interconnections, which can be problematic as changes can lead to assumptions about their influence including a lot of false positives, *i.e.* parts of the code which seem to be influenced, but are not.

For dynamic methods, execution traces are recorded during test case execution, giving a clearer image on what is actually affected by the test case. Dynamic call graphs [121], dynamic program slices [122] or control flow graphs [123] and data flow graphs [124] can be extracted from this information. While false positives are reduced, the found information is only valid for the executed scenarios. Also, in many cases code instrumentation is needed for recording the test execution to gather all needed information, which can alter the system's behavior as additional code is executed.

In contrast to traceability based CIA, dependency based CIA does not need additional manual modeling which facilitates applicability in production automation. Nevertheless, these approaches mostly developed in the domain of computer science have not been adapted in production automation. Only recently, first advances towards applying similar approaches in this domain were made [125].

The reason for the hesitant adaption are most likely rooted in the dominant programming standard IEC 61131-3. Even though there are many similarities between programming in the domains of computer science and production automation, most of the presented concepts are not directly applicable to automated production systems' program code: The IEC 61131-3 consists of different graphical as well as textual programming languages and exhibits differences regarding structure and behavior that make an adaptation complicated. Advances towards object-orientation are being made, yet the structure of IEC 61131-3 programs is neither completely object-oriented (e.g. many global accesses), nor procedural (e.g. class like function blocks). All of the analyzed approaches are directly aimed at one of these paradigms. Regarding behavior, cyclic execution of the code significantly influences the programming paradigm, and thus hinders a direct application of the approaches which usually assume a single execution of the program per test case. The test cases used in the publications are designed accordingly: single input vector test cases that do not allow adequate testing of state machines, which are common in automated production systems.

8. Discussion and Research Gaps

Reflecting the presented work in this paper, many promising approaches in the field of model-based testing in production automation have emerged.

Research Question 1: The automatic execution of test cases for testing IEC 61131-3 and PLC software applications is a field where a lot of work remains to be done. As mentioned in [6], full support of current tools for PLC platforms is still missing. Approaches that integrate automatic execution in available tools or new tools targeted at PLC platforms have scarcely been found.

Research Question 2: The same is true for change impact analysis and regression testing approaches which have mainly been researched in the field of computer science so far, as summarized in **Table 5**. Regarding traceability based CIA, the rare use of formalized models for functional specification definition in production automation hinders these approaches. Reasons for this can most likely be attributed to the uncertain relation of investment (software tools, personnel training, additional effort during the specification phase, etc.) and return (increased efficiency and software quality, etc.), yet detailed analyses are needed to find and approach the lead

[1]The expressions static and dynamic model generation are not to be confused with static and dynamic feedback inclusion in test cases.

causes for this situation. In dependency based CIA, adoption of the numerous approaches from computer science seems to be hindered by the programming standard IEC 61131-3, its programming languages, structure and properties regarding execution (cyclic, real time). A detailed investigation on the applicability and benefit of these methods in the field of production automation remains an open issue.

Research Question 3: Defining appropriate fault models and generating test cases to test the reaction to faults is still an emerging field in production automation, where mainly other domains have been conducting research until now. The use of FI to test hardware faults is widely spread to validate the dependability of integrated circuits. In this field all HWIFI, MIFI and SWIFI approaches have been tested and evaluated. The automotive and aerospace domains make use of the models available during the development process and introduce faults on model level. The execution is done using simulation or code-generation adopting the MIFI or SWIFI approach. In domains where models are scarcely available such as production automation, FI techniques on application level have not been exploited very much so far even though it is a field where reliability and dependability are of huge interest.

Research Question 4: Throughout the paper several HiL approaches in adjacent domain of production automation have been found. However, HiL approaches focusing on production automation remain scarce and have rarely been found.

Research Question 5: The usage of modeling languages that are applied in the engineering process—especially based on the UML—as a basis for test case generation and methods to derive test cases from these models have been investigated. However, comprehensive surveys and case studies on the acceptance and usability of these models or test case generation from SysML models in production automation still remain an open challenge. There are a number of approaches for transformation of established modeling languages into formal models for test case generation, as well as algorithms and test selection criteria. To find industrial relevance of the approaches, a thorough industrial evaluation could help assessing the relevance for the domain of production automation. A special focus should be given to approaches that include not only the source code but also at least the execution model of the implementation of the plant model.

9. Conclusion

In this paper, model-based testing approaches have been reviewed in context of the current challenges within the field of production automation. These challenges have been identified based on [2] as the generation of test cases applicable for PLC software, the minimization of testing efforts during regression testing, the inclusion of hardware effects and the generation of test-cases out of models which are accepted and applied within industry. Furthermore, the classification criteria in order to classify the different analyzed approaches are introduced, ordered by requirements and models as a basis for testing, test selection criteria, executable test cases, test bed, system under test and test verdict. In conclusion it has been found that, especially regarding the generation out of user-friendly notations as a basis for test case generation, testing of system's reaction to faults and regression testing, more work remains to be done. For the latter two, promising approaches have been introduced in electrical engineering and computer science which may be interesting for the field of production automation when adapted to the domain-specific requirements.

References

[1] (2010) ISO/IEC/IEEE 24765:2010(E), Systems and Software Engineering—Vocabulary.

[2] Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M. and Göhner, P. (2014) Challenges for Software Engineering in Automation. *Journal of Software Engineering and Applications*, **7**, 440-451. http://dx.doi.org/10.4236/jsea.2014.75041

[3] Wehrmeister, M.A., Pereira, C.E. and Rammig, F.J. (2013) Aspect-Oriented Model-Driven Engineering for Embedded Systems Applied to Automation Systems. *IEEE Transactions on Industrial Informatics*, **9**, 2373-2386. http://dx.doi.org/10.1109/TII.2013.2240308

[4] IEC 60880 (2006) Nuclear Power Plants—Instrumentation and Control Systems Important to Safety—Software Aspects for Computer-Based Systems Performing Category A Functions. International Electrotechnical Commission, 2nd Edition.

[5] IEC 61850-10 (2005) Communications Networks and Systems in Substations—Part 10: Conformance Testing. International Electrotechnical Commission, 2nd Edition.

[6] Dubey, A. (2011) Evaluating Software Engineering Methods in the Context of Automation Applications. 2011 9*th IEEE International Conference on Industrial Informatics* (*INDIN*), Caparica, 26-29 July 2011, 585-590. http://dx.doi.org/10.1109/INDIN.2011.6034944

[7] Utting, M., Pretschner, A. and Legeard, B. (2012) A Taxonomy of Model-Based Testing Approaches. *Software Testing, Verification and Reliability*, **22**, 297-312. http://dx.doi.org/10.1002/stvr.456

[8] Vyatkin, V. (2013) Software Engineering in Industrial Automation: State-of-the-Art Review. *IEEE Transactions on Industrial Informatics*, **9**, 1234-1249. http://dx.doi.org/10.1109/TII.2013.2258165

[9] Thramboulidis, K. and Frey, G. (2011) Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation. *Journal of Software Engineering and Applications*, **4**, 217-226. http://dx.doi.org/10.4236/jsea.2011.44024

[10] Corriveau, J.-P. and Shi, W. (2013) Traceability in Acceptance Testing. *Journal of Software Engineering and Applications*, **6**, 36-46. http://dx.doi.org/10.4236/jsea.2013.610A005

[11] Roussel, J.-M. and Lesage, J.-J. (1996) Validation and Verification of Grafcets Using Finite State Machine. *Proceedings of IMACS-IEEE, CESA*'96, Lille, 9-12 July 1996, 1-6.

[12] Frey, G. and Litz, L. (2000) Formal Methods in PLC Programming. 2000 *IEEE International Conference on Systems, Man, and Cybernetics*, **4**, 2431-2436. http://dx.doi.org/10.1109/icsmc.2000.884356

[13] Younis, M.B. and Frey, G. (2003) Formalization of Existing PLC Programs: A Survey. *Proceedings of IMACS-IEEE, CESA*'03, Lille, 9-11 July 2003, 234-239.

[14] Girbea, A., Suciu, C., Nechifor, S. and Sisak, F. (2014) Design and Implementation of a Service-Oriented Architecture for the Optimization of Industrial Applications. *IEEE Transactions on Industrial Informatics*, **10**, 185-196. http://dx.doi.org/10.1109/TII.2013.2253112

[15] Verein Deutscher Ingenieure (2005) Classification and Evaluation of Description Methods in Automation and Control Technology.

[16] Van Veenendaal, E. (2014) Standard Glossary of Terms Used in Software Testing. International Software Testing Qualifications Board, Version 2.3, 1-53.

[17] Hsueh, M.-C., Tsai, T.K. and Iyer, R.K. (1997) Fault Injection Techniques and Tools. *Computer*, **30**, 75-82. http://dx.doi.org/10.1109/2.585157

[18] Barth, M. and Fay, A. (2013) Automated Generation of Simulation Models for Control Code Tests. *Control Engineering Practice*, **21**, 218-230. http://dx.doi.org/10.1016/j.conengprac.2012.09.022

[19] Carlsson, H., Svensson, B., Danielsson, F. and Lennartson, B. (2012) Methods for Reliable Simulation-Based PLC Code Verification. *IEEE Transactions on Industrial Informatics*, **8**, 267-278. http://dx.doi.org/10.1109/TII.2011.2182653

[20] Gu, F., Harrison, W.S., Tilbury, D.M. and Yuan, C. (2007) Hardware-in-the-Loop for Manufacturing Automation Control: Current Status and Identified Needs. 2007 *IEEE International Conference on Automation Science and Engineering, CASE* 2007, Scottsdale, 22-25 September 2007, 1105-1110. http://dx.doi.org/10.1109/COASE.2007.4341787

[21] Myers, G.J., Sandler, C., Badgett, T. and Thomas, T.M. (2004) The Art of Software Testing. Second Edition, John Wiley & Sons, Hoboken.

[22] Thramboulidis, K. (2012) IEC 61499: Back to the Well Proven Practice of IEC 61131? 2012 *IEEE* 17*th Conference on Emerging Technologies & Factory Automation* (*ETFA*), Krakow, 17-21 September 2012, 1-8. http://dx.doi.org/10.1109/etfa.2012.6489672

[23] Zoitl, A. and Prähofer, H. (2013) Guidelines and Patterns for Building Hierarchical Automation Solutions in the IEC 61499 Modeling Language. *IEEE Transactions on Industrial Informatics*, **9**, 2387-2396. http://dx.doi.org/10.1109/TII.2012.2235449

[24] Avizienis, A., Laprie, J.-C., Randell, B. and Landwehr, C. (2004) Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, **1**, 11-33. http://dx.doi.org/10.1109/TDSC.2004.2

[25] Fernandez, J.-C., Jard, C., Jéron, T. and Viho, C. (1996) Using On-the-Fly Verification Techniques for the Generation of Test Suites. In: Alur, R. and Henzinger, T.A., Eds., *Computer Aided Verification*, Springer, Berlin, 348-359. http://dx.doi.org/10.1007/3-540-61474-5_82

[26] Lee, D. and Yannakakis, M. (1996) Principles and Methods of Testing Finite State Machines—A Survey. *Proceedings of the IEEE*, **84**, 1090-1123. http://dx.doi.org/10.1109/5.533956

[27] Tretmans, J. (1996) Test Generation with Inputs, Outputs and Repetitive Quiescence. *Software, Concepts and Tools*, **17**, 103-120.

[28] Tretmans, J. (2008) Model Based Testing with Labelled Transition Systems. In: Hierons, R.M., Bowen, J.P. and Har-

man, M., Eds., *Formal Methods and Testing*, Lecture Notes in Computer Science, Vol. 4949, Springer, Berlin, 1-38. http://dx.doi.org/10.1007/978-3-540-78917-8_1

[29] Pickin, S., Jard, C., Jéron, T., Jézéquel, J.-M. and Le Traon, Y. (2007) Test Synthesis from UML Models of Distributed Software. *IEEE Transactions on Software Engineering*, **33**, 252-269. http://dx.doi.org/10.1109/TSE.2007.39

[30] Krichen, M. and Tripakis, S. (2009) Conformance Testing for Real-Time Systems. *Formal Methods in System Design*, **34**, 238-304. http://dx.doi.org/10.1007/s10703-009-0065-1

[31] Pocci, M., Demongodin, I., Giambiasi, N. and Giua, A. (2014) Testing Experiments on Synchronized Petri Nets. *IEEE Transactions on Automation Science and Engineering*, **11**, 125-138. http://dx.doi.org/10.1109/TASE.2013.2290774

[32] Guignard, A. and Faure, J.-M. (2014) A Conformance Relation for Model-Based Testing of PLC. *Proceedings of the* 12*th International Workshop on Discrete Event Systems-WODES* 2014, Cachan, 14-16 May 2014, 412-419.

[33] Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M. and Ghedamsi, A. (1991) Test Selection Based on Finite State Models. *IEEE Transactions on Software Engineering*, **17**, 591-603. http://dx.doi.org/10.1109/32.87284

[34] Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A.R. and Yevtushenko, N. (2010) FSM-Based Conformance Testing Methods: A Survey Annotated with Experimental Evaluation. *Information and Software Technology*, **52**, 1286-1297. http://dx.doi.org/10.1016/j.infsof.2010.07.001

[35] Endo, A.T. and Simao, A. (2013) Evaluating Test Suite Characteristics, Cost, and Effectiveness of FSM-Based Testing Methods. *Information and Software Technology*, **55**, 1045-1062. http://dx.doi.org/10.1016/j.infsof.2013.01.001

[36] Petrenko, A., Simao, A. and Yevtushenko, N. (2012) Generating Checking Sequences for Nondeterministic Finite State Machines. 2012 *IEEE* 5*th International Conference on Software Testing, Verification and Validation* (*ICST*), Montreal, 17-21 April 2012, 310-319. http://dx.doi.org/10.1109/ICST.2012.111

[37] Hierons, R.M. and Türker, U.C. (2014) Distinguishing Sequences for Partially Specified FSMs. In: Badger, J.M. and Rozier, K.Y., Eds., *NASA Formal Methods*, Springer, Cham, 62-76. http://dx.doi.org/10.1007/978-3-319-06200-6_5

[38] Pocci, M., Demongodin, I., Giambiasi, N. and Giua, A. (2013) A New Algorithm to Compute Synchronizing Sequences for Synchronized Petri Nets. 2013 *IEEE Region* 10 *Conference* (31194), *TENCON* 2013, Xi'an, 22-25 October 2013, 1-6. http://dx.doi.org/10.1109/tencon.2013.6718970

[39] Kaner, C., Bach, J. and Pettichord, B. (2008) Lessons Learned in Software Testing. John Wiley & Sons, Hoboken.

[40] Simão, A., Petrenko, A. and Yevtushenko, N. (2009) Generating Reduced Tests for FSMs with Extra States. In: Núñez, M., Baker, P. and Merayo, M.G., Eds., *Testing of Software and Communication Systems*, Springer, Berlin, 129-145. http://dx.doi.org/10.1007/978-3-642-05031-2_9

[41] El-Fakih, K., Yevtushenko, N. and Fouchal, H. (2009) Testing Timed Finite State Machines with Guaranteed Fault Coverage. In: Núñez, M., Baker, P. and Merayo, M.G., Eds., *Testing of Software and Communication Systems*, Springer, Berlin, 66-80. http://dx.doi.org/10.1007/978-3-642-05031-2_5

[42] White, L.J. and Cohen, E.I. (1980) A Domain Strategy for Computer Program Testing. *IEEE Transactions on Software Engineering*, **SE-6**, 247-257. http://dx.doi.org/10.1109/TSE.1980.234486

[43] Andrade, W.D.L., Machado, P.D., Jeron, T. and Marchand, H. (2011) Abstracting Time and Data for Conformance Testing of Real-Time Systems. *IEEE* 4*th International Conference on Software Testing, Verification and Validation Workshops* (*ICSTW*), Berlin, 21-25 March 2011, 9-17. http://dx.doi.org/10.1109/icstw.2011.82

[44] Andrade, W.D.L. and Machado, P.D. (2013) Generating Test Cases for Real-Time Systems Based on Symbolic Models. *IEEE Transactions on Software Engineering*, **39**, 1216-1229. http://dx.doi.org/10.1109/TSE.2013.13

[45] Cadar, C. and Sen, K. (2013) Symbolic Execution for Software Testing: Three Decades Later. *Communications of the ACM*, **56**, 82-90. http://dx.doi.org/10.1145/2408776.2408795

[46] Petrenko, A. and Yevtushenko, N. (2005) Testing from Partial Deterministic FSM Specifications. *IEEE Transactions on Computers*, **54**, 1154-1165. http://dx.doi.org/10.1109/TC.2005.152

[47] Shahbaz, M. and Groz, R. (2014) Analysis and Testing of Black-Box Component-Based Systems by Inferring Partial Models. *Software Testing, Verification and Reliability*, **24**, 253-288. http://dx.doi.org/10.1002/stvr.1491

[48] Constant, C., Jeron, T., Marchand, H. and Rusu, V. (2007) Integrating Formal Verification and Conformance Testing for Reactive Systems. *IEEE Transactions on Software Engineering*, **33**, 558-574. http://dx.doi.org/10.1109/TSE.2007.70707

[49] Armando, A., Pellegrino, G., Carbone, R., Merlo, A. and Balzarotti, D. (2012) From Model-Checking to Automated Testing of Security Protocols: Bridging the Gap. In: Brucker, A.D. and Julliand, J., Eds., *Tests and Proofs*, Springer, Berlin, 3-18. http://dx.doi.org/10.1007/978-3-642-30473-6_3

[50] Enoiu, E.P., Sundmark, D. and Pettersson, P. (2013) Model-Based Test Suite Generation for Function Block Diagrams Using the UPPAAL Model Checker. 2013 *IEEE* 6th *International Conference on Software Testing, Verification and Validation Workshops* (*ICSTW*), Luxembourg, 18-22 March 2013, 158-167. http://dx.doi.org/10.1109/ICSTW.2013.27

[51] Gaudel, M.-C. (2011) Checking Models, Proving Programs, and Testing Systems. In: Gogolla, M. and Wolff, B., Eds., *Tests and Proofs*, Springer, Berlin, 1-13. http://dx.doi.org/10.1007/978-3-642-21768-5_1

[52] Gaudel, M.-C., Lassaigne, R., Magniez, F. and de Rougemont, M. (2013) Some Approximations in Model Checking and Testing. arXiv: 1304.5199.

[53] Hanisch, H.-M., Thieme, J., Luder, A. and Wienhold, O. (1997) Modeling of PLC Behavior by Means of Timed Net Condition/Event Systems. 1997 6*th International Conference on Emerging Technologies and Factory Automation Proceedings*, *ETFA*'97, Los Angeles, 9-12 September 1997, 391-396. http://dx.doi.org/10.1109/ETFA.1997.616302

[54] Heiner, M. and Menzel, T. (1998) A Petri Net Semantics for the PLC Language Instruction List. 4*th Workshop on Discrete Event Systems* (*WODES*'98), Cagliari, 26-28 August 1998, 161-166.

[55] Mader, A. and Wupper, H. (1999) Timed Automaton Models for Simple Programmable Logic Controllers. *Proceedings of the* 11*th Euromicro Conference on Real-Time Systems*, York, 9-11 June 1999, 106-113. http://dx.doi.org/10.1109/emrts.1999.777456

[56] Canet, G., Couffin, S., Lesage, J.-J., Petit, A. and Schnoebelen, P. (2000) Towards the Automatic Verification of PLC Programs Written in Instruction List. *2000 IEEE International Conference on Systems, Man, and Cybernetics*, **4**, 2449-2454. http://dx.doi.org/10.1109/icsmc.2000.884359

[57] Jiménez-Fraustro, F. and Rutten, E. (2001) A Synchronous Model of IEC 61131 PLC Languages in SIGNAL. 13*th Euromicro Conference on Real-Time Systems*, Delft, 13-15 June 2001, 135-142. http://dx.doi.org/10.1109/EMRTS.2001.934016

[58] Gourcuff, V., De Smet, O. and Faure, J. (2006) Efficient Representation for Formal Verification of PLC Programs. 2006 8*th International Workshop on Discrete Event Systems*, Ann Arbor, 10-12 July 2006, 182-187. http://dx.doi.org/10.1109/WODES.2006.1678428

[59] Zoubek, B., Roussel, J.-M. and Kwiatkowska, M. (2003) Towards Automatic Verification of Ladder Logic Programs. *Proceedings of IMACS-IEEE CESA*'03: *Computational Engineering in Systems Applications*, Lille, 9-11 July 2003.

[60] Bender, D.F., Combemale, B., Crégut, X., Farines, J.M., Berthomieu, B. and Vernadat, F. (2008) Ladder Metamodeling and PLC Program Validation through Time Petri Nets. In: Schieferdecker, I. and Hartman, A., Eds., *Model Driven Architecture—Foundations and Applications*, Springer, Berlin, 121-136. http://dx.doi.org/10.1007/978-3-540-69100-6_9

[61] Bel Mokadem, H., Berard, B., Gourcu, V., De Smet, O. and Roussel, J.-M. (2010) Verification of a Timed Multitask System with UPPAAL. *IEEE Transactions on Automation Science and Engineering*, **7**, 921-932. http://dx.doi.org/10.1109/TASE.2010.2050199

[62] Pavlovic, O. and Ehrich, H.-D. (2010) Model Checking PLC Software Written in Function Block Diagram. 2010 3*rd International Conference on Software Testing, Verification and Validation* (*ICST*), Paris, 6-10 April 2010, 439-448. http://dx.doi.org/10.1109/icst.2010.10

[63] Soliman, D., Thramboulidis, K. and Frey, G. (2012) Transformation of Function Block Diagrams to UPPAAL Timed Automata for the Verification of Safety Applications. *Annual Reviews in Control*, **36**, 338-345. http://dx.doi.org/10.1016/j.arcontrol.2012.09.015

[64] Wardana, A., Folmer, J. and Vogel-Heuser, B. (2009) Automatic Program Verification of Continuous Function Chart Based on Model Checking. 35*th Annual Conference of IEEE Industrial Electronics*, Porto, 3-5 November 2009, 2422-2427. http://dx.doi.org/10.1109/iecon.2009.5415231

[65] Yoong, L.H., Roop, P.S., Vyatkin, V. and Salcic, Z. (2009) A Synchronous Approach for IEC 61499 Function Block Implementation. *IEEE Transactions on Computers*, **58**, 1599-1614. http://dx.doi.org/10.1109/TC.2009.128

[66] Prähofer, H. and Zoitl, A. (2013) Verification of Hierarchical IEC 61499 Component Systems with Behavioral Event Contracts. 2013 11*th IEEE International Conference on Industrial Informatics* (*INDIN*), Bochum, 29-31 July 2013, 578-585. http://dx.doi.org/10.1109/INDIN.2013.6622948

[67] L'Her, D., Le Parc, P. and Marce, L. (1999) Proving Sequential Function Chart Programs Using Automata. In: Champarnaud, J.-M., Ziadi, D. and Maurel, D., Eds., *Automata Implementation*, Springer, Berlin, 149-163. http://dx.doi.org/10.1007/3-540-48057-9_13

[68] Remelhe, M., Lohmann, S., Stursberg, O., Engell, S. and Bauer, N. (2004) Algorithmic Verification of Logic Controllers Given as Sequential Function Charts. 2004 *IEEE International Symposium on Computer Aided Control Systems Design*, Taipei, 4 September 2004, 53-58. http://dx.doi.org/10.1109/CACSD.2004.1393850

[69] Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M. and Stursberg, O. (2004) Verification of PLC Programs Given as Sequential Function Charts. In: Ehrig, H., Damm, W., Desel, J., Große-Rhode, M., Reif, W., Schnieder, E. and Westkämper, E., Eds., *Integration of Software Specification Techniques for Applications in Engineering*, Lecture Notes in Computer Science, Vol. 3147, Part V: Verification, Springer, Berlin, 517-540. http://dx.doi.org/10.1007/978-3-540-27863-4_28

[70] Bauer, N., Huuck, R., Lukoschus, B. and Engell, S. (2004) A Unifying Semantics for Sequential Function Charts. In: Ehrig, H., Damm, W., Desel, J., Große-Rhode, M., Reif, W., Schnieder, E., Westkämper, E., Eds., *Integration of Software Specification Techniques for Applications in Engineering*, Springer, Berlin, 400-418. http://dx.doi.org/10.1007/978-3-540-27863-4_22

[71] Provost, J., Roussel, J.-M. and Faure, J.-M. (2011) Translating Grafcet Specifications into Mealy Machines for Conformance Test Purposes. *Control Engineering Practice*, **19**, 947-957. http://dx.doi.org/10.1016/j.conengprac.2010.10.001

[72] Provost, J., Roussel, J.-M. and Faure, J.-M. (2011) A Formal Semantics for Grafcet Specifications. *Proceedings of the IEEE 7th International Conference on Automation Science and Engineering* (*CASE* 2011), Trieste, 24-27 August 2011, 488-494. http://dx.doi.org/10.1109/CASE.2011.6042457

[73] Provost, J., Roussel, J.-M. and Faure, J.-M. (2014) Generation of Single Input Change Test Sequences for Conformance Test of Programmable Logic Controllers. *IEEE Transactions on Industrial Informatics*, **10**, 1696-1704. http://dx.doi.org/10.1109/TII.2014.2315972

[74] Schumacher, F., Schröck, S. and Fay, A. (2013) Transforming Hierarchical Concepts of GRAFCET into a Suitable Petri Net Formalism. *Manufacturing Modelling, Management, and Control*, **7**, 295-300.

[75] Schumacher, F. and Fay, A. (2013) Transforming Time Constraints of a GRAFCET Graph into a Suitable Petri Net Formalism. 2013 *IEEE International Conference on Industrial Technology* (*ICIT*), Cape Town, 25-28 February 2013, 210-218. http://dx.doi.org/10.1109/ICIT.2013.6505674

[76] Hamon, G. and Rushby, J. (2007) An Operational Semantics for Stateflow. *International Journal on Software Tools for Technology Transfer*, **9**, 447-456. http://dx.doi.org/10.1007/s10009-007-0049-7

[77] Beer, A., Georgiev, T., Leitner-Fischer, F. and Leue, S. (2013) Model-Based Quantitative Safety Analysis of Matlab Simulink/Stateow Models. *Tagungsband des Dagstuhl-Workshops*, Dagstuhl, 24-26 April 2013.

[78] Jouault, F., Allilaire, F., Bezivin, J. and Kurtev, I. (2008) ATL: A Model Transformation Tool. *Science of Computer Programming*, **72**, 31-39. http://dx.doi.org/10.1016/j.scico.2007.08.002

[79] Schetinin, N., Moriz, N., Kumar, B., Maier, A., Faltinski, S. and Niggemann, O. (2013) Why Do Verification Approaches in Automation Rarely Use HIL-Test? 2013 *IEEE International Conference on Industrial Technology* (*ICIT*), Cape Town, 25-28 February 2013, 1428-1433. http://dx.doi.org/10.1109/ICIT.2013.6505881

[80] Obermeier, M., Braun, S. and Vogel-Heuser, B. (2014) A Model-Driven Approach on Object Oriented PLC Programming for Manufacturing Systems with Regard to Usability. *IEEE Transactions on Industrial Informatics*, **11**, 790-800. http://dx.doi.org/10.1109/TII.2014.2346133

[81] Hametner, R., Winkler, D., Östreicher, T., Bi, S. and Zoitl, A. (2010) The Adaptation of Test-Driven Software Processes to Industrial Automation Engineering. 2010 8*th IEEE International Conference on Industrial Informatics* (*INDIN*), Osaka, 13-16 July 2010, 921-927. http://dx.doi.org/10.1109/indin.2010.5549620

[82] Hussain, T. and Frey, G. (2006) UML-Based Development Process for IEC 61499 with Automatic Test-Case Generation. *IEEE Conference on Emerging Technologies and Factory Automation* (*ETFA*'06), Prague, 20-22 September 2006, 1277-1284. http://dx.doi.org/10.1109/etfa.2006.355407

[83] Hametner, R., Kormann, B., Vogel-Heuser, B., Winkler, D. and Zoitl, A. (2011) Test Case Generation Approach for Industrial Automation Systems. 2011 5*th International Conference on Automation, Robotics and Applications*, Wellington, 6-8 December 2011, 57-62. http://dx.doi.org/10.1109/ICARA.2011.6144856

[84] Krause, J., Herrmann, A. and Diedrich, C. (2008) Test Case Generation from Formal System Specifications Based on UML State Machine. *ATP-International*, **1**, 47-54.

[85] Kormann, B., Tikhonov, D. and Vogel-Heuser, B. (2012) Automated PLC Software Testing Using Adapted UML Sequence Diagrams. 14*th IFAC Symposium of Information Control Problems in Manufacturing*, **14**, 1615-1621.

[86] Zander, J., Dai, Z.R., Schieferdecker, I. and Din, G. (2005) From U2TP Models to Executable Tests with TTCN-3—An Approach to Model Driven Testing. In: Khendek, F. and Dssouli, R., Eds., *Testing of Communicating Systems*, Springer, Berlin, 289-303. http://dx.doi.org/10.1007/11430230_20

[87] Kumar, B., Czybik, B. and Jasperneite, J. (2011) Model Based TTCN-3 Testing of Industrial Automation Systems—First Results. 2011 *IEEE* 16*th Conference on Emerging Technologies and Factory Automation*, Toulouse, 5-9 September 2011, 1-4. http://dx.doi.org/10.1109/etfa.2011.6059146

[88] DeTommasi, G., Vitelli, R., Boncagni, L. and Neto, A.C. (2013) Modeling of MARTe-Based Real-Time Applications with SysML. *IEEE Transactions on Industrial Informatics*, **9**, 2407-2415. http://dx.doi.org/10.1109/TII.2012.2235073

[89] Bayrak, G. and Vogel-Heuser, B. (2014) Evaluation of Programming Languages for a Flexible Programming of Thermo-Mechanical Processes by Means of Cognitive Effectiveness. *IEEE*.

[90] Bayrak, G., Murr, P., Ulewicz, S. and Vogel-Heuser, B. (2012) Comparison of a Transformed Matlab/Simulink Model

into the Programming Language CFC on Different IEC 61131-3 PLC Environments. 2012 *IEEE* 17*th Conference on Emerging Technologies & Factory Automation* (*ETFA*), Krakow, 17-21 September 2012, 1-8. http://dx.doi.org/10.1109/ETFA.2012.6489667

[91] Petrenko, A., Dury, A., Ramesh, S. and Mohalik, S. (2013) A Method and Tool for Test Optimization for Automotive Controllers. 2013 *IEEE* 6*th International Conference on Software Testing, Verification and Validation Workshops* (*ICSTW*), Luxembourg, 18-22 March 2013, 198-207. http://dx.doi.org/10.1109/ICSTW.2013.31

[92] Zelenov, S., Silakov, D., Petrenko, A., Conrad, M. and Fey, I. (2006) Automatic Test Generation for Model-Based Code Generators. *ISoLA* 2006, 2*nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, Paphos, 15-19 November 2006, 75-81. http://dx.doi.org/10.1109/ISoLA.2006.70

[93] Estévez, E. and Marcos, M. (2012) Model-Based Validation of Industrial Control Systems. *IEEE Transactions on Industrial Informatics*, **8**, 302-310. http://dx.doi.org/10.1109/TII.2011.2174248

[94] Svenningsson, R., Vinter, J., Eriksson, H. and Törngren, M. (2010) MODIFI: A MODel-Implemented Fault Injection Tool. In: Schoitsch, E., Ed., *Computer Safety, Reliability, and Security*, Lecture Notes in Computer Science, Vol. 6351, Springer, Berlin, 210-222. http://dx.doi.org/10.1007/978-3-642-15651-9_16

[95] Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.-C., Laprie, J.-C., Martins, E. and Powell, D. (1990) Fault Injection for Dependability Validation: A Methodology and Some Applications. *IEEE Transactions on Software Engineering*, **16**, 166-182. http://dx.doi.org/10.1109/32.44380

[96] Aidemark, J., Vinter, J., Folkesson, P. and Karlsson, J. (2001) GOOFI: Generic Object-Oriented Fault Injection Tool. 2001 *International Conference on Dependable Systems and Networks, DSN* 2001, Goteborg, 1-4 July 2001, 83-88. http://dx.doi.org/10.1109/DSN.2001.941394

[97] Ziade, H., Ayoubi, R. and Velazco, R. (2004) A Survey on Fault Injection Techniques. *The International Arab Journal of Information Technology*, **1**, 171-186.

[98] Baraza, J., Gracia, J., Gil, D. and Gil, P. (2005) Improvement of Fault Injection Techniques Based on VHDL Code Modification. 10*th IEEE International High-Level Design Validation and Test Workshop*, 30 November-2 December 2005, 19-26. http://dx.doi.org/10.1109/hldvt.2005.1568808

[99] Schlingloff, H. and Vulinovic, S. (2005) Model Based Dependability Evaluation for Automotive Control Functions. In: Fritzson, P., Ed., *Modeling and Simulation for Public Safety*.

[100] Vinterl, J., Bromander, L., Raistrick, P. and Edlerl, H. (2007) FISCADE—A Fault Injection Tool for SCADE Models. 2007 3*rd Institution of Engineering and Technology Conference on Automotive Electronics*, Warwick, 28-29 June 2007, 1-9.

[101] Powell, D., Arlat, J., Chu, H.N., Ingrand, F. and Killijian, M. (2012) Testing the Input Timing Robustness of Real-Time Control Software for Autonomous Systems. 2012 9*th European Dependable Computing Conference*, Sibiu, 8-11 May 2012, 73-83. http://dx.doi.org/10.1109/EDCC.2012.16

[102] Kormann, B. and Vogel-Heuser, B. (2011) Automated Test Case Generation Approach for PLC Control Software Exception Handling Using Fault Injection. *IECON* 2011—37*th Annual Conference on IEEE Industrial Electronics Society*, Melbourne, 7-10 November 2011, 365-372. http://dx.doi.org/10.1109/IECON.2011.6119280

[103] Rösch, S., Tikhonov, D., Schütz, D. and Vogel-Heuser, B. (2014) Model-Based Testing of PLC Software: Test of Plants' Reliability by Using Fault Injection on Component Level. *IFAC World Conference*, Cape Town, 24-29 August 2014.

[104] Sung, A., Choi, B., Wong, W.E. and Debroy, V. (2011) Mutant Generation for Embedded Systems Using Kernel-Based Software and Hardware Fault Simulation. *Information and Software Technology*, **53**, 1153-1164. http://dx.doi.org/10.1016/j.infsof.2011.03.010

[105] Arlat, J. and Crouzet, Y. (2010) Physical Fault Models and Fault Tolerance. In: Wunderlich, H.-J., Ed., *Models in Hardware Testing*, Frontiers in Electronic Testing, Vol. 43, Springer Netherlands, Dordrecht, 217-255. http://dx.doi.org/10.1007/978-90-481-3282-9_8

[106] Schludermann, H., Kirchmair, T. and Vorderwinkler, M. (2000) Soft-Commissioning: Hardware-in-the-Loop-Based Verification of Controller Software. *Proceedings of the* 2000 *Winter Simulation Conference*, **1**, 893-899. http://dx.doi.org/10.1109/wsc.2000.899889

[107] Carreira, J., Madeira, H. and Silva, J.G. (1998) Xception: Software Fault Injection and Monitoring in Processor Functional Units. *Dependable Computing and Fault Tolerant Systems*, **10**, 245-266.

[108] Yuste, P., Ruiz, J.C., Lemus, L. and Gil, P. (2003) Non-Intrusive Software-Implemented Fault Injection in Embedded Systems. In: de Lemos, R., Weber, T.S. and Camargo Jr., J.B., Eds., *Dependable Computing*, Springer, Berlin, 23-38. http://dx.doi.org/10.1007/978-3-540-45214-0_5

[109] (1998) IEEE Std 1219-1998, IEEE Standard for Software Maintenance. The Institute of Electrical and Electronics Engineers, Inc., Los Alamos.

[110] Rothermel, G. and Harrold, M. (1996) Analyzing Regression Test Selection Techniques. *IEEE Transactions on Software Engineering*, **22**, 529-551. http://dx.doi.org/10.1109/32.536955

[111] Yoo, S. and Harman, M. (2010) Regression Testing Minimization, Selection and Prioritization: A Survey. *Software Testing, Verification and Reliability*, **22**, 67-120. http://dx.doi.org/10.1002/stv.430

[112] Bohner, S.A. and Arnold, R.S. (1996) Software Change Impact Analysis. The Institute of Electrical and Electronic Engineers, Inc., Los Alamos.

[113] Farooq, Q., Iqbal, M.Z.Z., Malik, Z.I. and Nadeem, A. (2007) An Approach for Selective State Machine Based Regression Testing. *Proceedings of the* 3*rd International Workshop on Advances in Model-Based Testing* (A-MOST'07), ACM Press, New York, 44-52. http://dx.doi.org/10.1145/1291535.1291540

[114] Briand, L., Labiche, Y. and Soccar, G. (2002) Automating Impact Analysis and Regression Test Selection Based on UML Designs. *International Conference on Software Maintenance*, Montreal, 3-6 October 2002, 252-261. http://dx.doi.org/10.1109/icsm.2002.1167775

[115] Le Traon, Y., Jéron, T., Jézéquel, J.-M. and Morel, P. (2000) Efficient Object-Oriented Integration and Regression Testing. *IEEE Transactions on Reliability*, **49**, 12-25. http://dx.doi.org/10.1109/24.855533

[116] Caliebe, P., Herpel, T. and German, R. (2012) Dependency-Based Test Case Selection and Prioritization in Embedded Systems. 2012 *IEEE* 5*th International Conference on Software Testing, Verification and Validation*, Montreal, 17-21 April 2012, 731-735. http://dx.doi.org/10.1109/ICST.2012.164

[117] Chen, Y., Probert, R.L. and Ural, H. (2009) Regression Test Suite Reduction Based on SDL Models of System Requirements. *Journal of Software Maintenance and Evolution*: *Research and Practice*, **21**, 379-405. http://dx.doi.org/10.1002/smr.415

[118] Korel, B., Tahat, L. and Vaysburg, B. (2002) Model Based Regression Test Reduction Using Dependence Analysis. 2002 *International Conference on Software Maintenance*, Montreal, 3-6 October 2002, 214-223. http://dx.doi.org/10.1109/icsm.2002.1167768

[119] Ryder, B. and Tip, F. (2001) Change Impact Analysis for Object-Oriented Programs. *Proceedings of the* 2001 *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, 46-53. http://dx.doi.org/10.1145/379605.379661

[120] Leung, H. and White, L. (1990) A Study of Integration Testing and Software Regression at the Integration Level. 1990 *International Conference on Software Maintenance*, San Diego, 26-29 November 1990, 290-301. http://dx.doi.org/10.1109/icsm.1990.131377

[121] Ren, X., Shah, F., Tip, F., Ryder, B.G. and Chesley, O. (2004) Chianti: A Tool for Change Impact Analysis of Java Programs Categories and Subject Descriptors. *ACM Sigplan Notices*, **39**, 432-448. http://dx.doi.org/10.1145/1028976.1029012

[122] Orso, A., Apiwattanapong, T. and Harrold, M.J. (2003) Leveraging Field Data for Impact Analysis and Regression Testing. *ACM SIGSOFT Software Engineering Notes*, **28**, 128-137. http://dx.doi.org/10.1145/949952.940089

[123] Rothermel, G. and Harrold, M.J. (1997) A Safe, Efficient Regression Test Selection Technique. *ACM Transactions on Software Engineering and Methodology*, **6**, 173-210. http://dx.doi.org/10.1145/248233.248262

[124] Chen, Y.F., Rosenblum, D. and Vo, K. (1994) TESTTUBE: A System for Selective Regression Testing. *Proceedings of* 16*th International Conference on Software Engineering*, (*ICSE*'94), Sorrento, 16-21 May 1994, 211-220. http://dx.doi.org/10.1109/ICSE.1994.296780

[125] Ulewicz, S., Schütz, D. and Vogel-Heuser, B. (2014) Software Changes in Factory Automation—Towards Automatic Change Based Regression Testing. *IECON* 2014, 40*th Annual Conference of the IEEE Industrial Electronics Society*, Dallas, 29 October-1 November 2014, 2617-2623. http://dx.doi.org/10.1109/IECON.2014.7048875