Enabling Functional Integration Testing of Software-Intensive Technical Systems by Heterogeneous Models

Thomas Bauer, Frank Elberzhager
Fraunhofer Institute for Experimental Software Engineering IESE
Kaiserslautern, Germany
{thomas.bauer | frank.elberzhager}@iese.fraunhofer.de

Abstract— **In complex software-intensive systems, the analytical quality assurance activities on different levels have become crucial for achieving high product quality. Higher complexity and distributed product development require systematic integration testing to assure interoperability between components and the fulfilment of complex distributed system operations. This work presents the novel automated model-based testing approach ER!S for software-intensive technical systems, which uses a heterogeneous modeling concept for describing the test- and system-specific information. Recommendations from the relevant process standards have been considered to assure and support industrial applicability. The generic approach has been instantiated for functional integration testing on the software design level. It focuses on the functional requirements that are related to distributed system operations implemented by the component interplay. The test model contains the information needed for deriving the test cases for concrete stimulation sequences together with the corresponding expected behavior. The approach supports stepwise system assembly according to an operation-oriented integration strategy. The approach has been initially evaluated in a feasibility study, which was conducted in a research project together with tool vendors and industrial partners from different technical system domains. The first evaluation results are presented. A higher degree of test coverage regarding the relevant functional requirements was achieved.**

Keywords— model-based testing; software integration testing; standard-compliant quality assurance; ISO 26262.

I. INTRODUCTION

The increasing use of software in technical devices like automotive and aerospace systems has enabled the efficient development of new functionality. Software-intensive systems are the main innovation drivers for many embedded system domains nowadays. Most of the innovations are achieved by embedded software [3]. The increasing complexity of software-intensive technical systems regarding their functionality, requirements, system structure, and amount of program code requires more constructive and analytical measures to fulfill the high quality needs within the given economic limits [4].

One consequence of the ever greater complexity of systems and their software parts is the increasing impact of software defects on the overall system quality [2]. A significant number of defects are caused by the faulty interplay of software-controlled components that perform complex functions, the so-called distributed system operations. Therefore, integration and interoperability testing of distributed systems are essential quality assurance activities to check complex sub-system requirements, distributed system operations, and component interaction patterns.

In the research project MBAT, which stands for combined model-based analysis and testing [1], we investigate and develop quality assurance (QA) techniques for safety-related systems from the automotive, avionics, and rail domains.

Development and QA of these systems is guided and driven by different process standards depending on the application domain, e.g., ISO 26262 [6] for passenger cars and DO-178C [7] for airborne systems. Compliance of processes with such standards is an important factor that has to be considered when new technologies are introduced that tackle the challenges of increasing product complexity and economic restrictions.

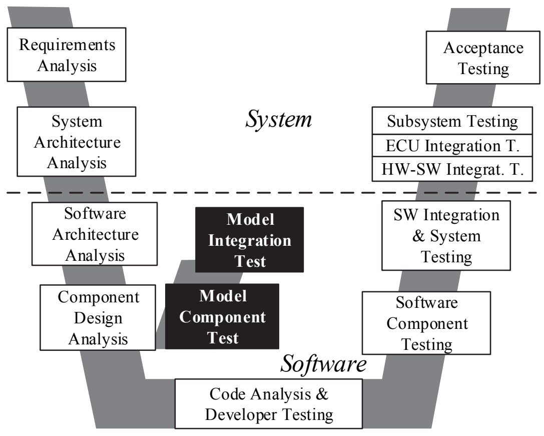
Requirements Analysis
System Architecture Analysis
Software Architecture Analysis
Component Design Analysis
System
Model Integration Test
Model Component Test
Software
Code Analysis & Developer Testing
Acceptance Testing
Subsystem Testing
ECU Integration T.
HW-SW Integrat. T.
SW Integration & System Testing
Software Component Testing

Figure 1. Simplified QA process for software-intensive technical systems

Figure 1 shows a simplified QA process for software-intensive technical systems. Test objects are executable software, software integrated with hardware, and networks of control units driven by software, which leads to various stages of integration testing. Software testing is split into several abstraction levels such as software component, integration, and system testing, where individual components, interacting subsystems, and fully integrated parts are checked against their specifications. If executable models are available from the design stages, dedicated model testing activities on the component and subsystem levels are conducted in addition. This leads to a new branch in the QA process, which is marked in black in the figure.

This article presents the new test approach ER!S for integration testing on the software and model levels of technical software-based systems. It is structured as follows: Section II presents the results from a state-of-the-practice study, which comprises a detailed analysis of the relevant process standards. Section III gives an overview and assessment of the state-of-the-art approaches in model-based integration testing and motivates why the available solutions are not sufficient. The new model-based testing approach ER!S is presented in Section IV (modeling notation) and Section V (test case generation approach). The article concludes with a short presentation of the evaluation results in Section VI and a summary in Section VII.

II. Analysis of Process Standards

Since technical software-intensive systems perform safety-related functions, standards and guidelines have been developed that are mandatory for product development and QA. Standards provide a high-level overview of the accepted and determined state of the practice at a defined time. They require compliance of the development processes regarding the activities performed and the documents produced depending on criticality degrees. The relevant standards for software development and verification in the transportation domains are IEC 61508 (generic recommendations for safety-related software-intensive systems, [5]), ISO 26262 (for passenger cars, [6]), DO-178C (for avionics systems, [7]), and EN 50128 (for the railway domain, [8]). In our analysis, we focused on ISO 26262 and DO-178C. ISO 26262 re-uses many recommendations and guidelines from the generic standard IEC 61508, which will not be considered separately in this section. The same applies to EN50218, which does not provide additional information for the state-of-the-practice analysis.

Software integration testing is explicitly considered and demanded in ISO 26262 and DO-178C as a mandatory QA activity. ISO 26262 states a set of objectives, coverage criteria, and test case derivation methods. DO-178C and its supplements for model-based development and verification (DO-331) and formal methods (DO-333) define a list of generic objectives to be satisfied during the development and QA of the different artifacts.

Figure 8 in the appendix section shows the relevant recommendations from the two major standards analyzed and the derived and aggregated recommendations focusing on functional software integration testing. The left part of the figure describes the DO-178C recommendations and the right part covers the recommendations from ISO 26262. The generic and aggregated conclusions are described at the bottom of the figure. The corresponding parts, sections, and tables in the documents and the external sources that are referenced in standards are annotated.

Certain recommendations depend on the safety criticality of the artifacts being developed and verified. A higher level of criticality always leads to stricter recommendations and more intensive QA. Both standards have four criticality levels (*A, B, C, D*), but with different orders. The most critical ISO level is D and the highest DO level is A.

ISO 26262 distinguishes between recommended and mandatory actions, which are annotated as lower and upper case letters in the figure. For example, the annotation *abCD* of the ISO recommendation *function coverage* means that this criterion is recommended for criticality levels A and B and mandatory for C and D. The term (*--BA)* for the DO recommendation of the criterion *branch coverage of source code* means that this criterion is recommended for levels A and B, but not needed for levels C and D. Directed arrows show the references between different document parts. Dashed lines represent the relations of the documents and sections of the standards to the generic and aggregated recommendations at the bottom of the figure.

The major recommendation of both standards is the intensive verification and validation of a product's compliance with its requirements. DO-178C, in particular, demands very strict approaches for requirements refinement, traceability, and coverage in the test cases. Additional aspects cover verification of compliance with the software architecture design and the interface definitions, performance properties checks, and coverage of exceptional situations in the robustness test.

Concrete techniques for test case selection are also proposed. Both standards mention the generic, industrially proven, functional testing techniques of equivalence class partitioning, boundary value analysis, and coverage of specification and design models if such model models are available. Detailed criteria and guidelines for the application of these techniques are not provided. An exception is the recommendation of model coverage for finite state machines. The definition of concrete equivalence classes and boundary values and their exploitation for the selection of test cases are not further defined and remain up to the test designer.

Furthermore, architectural coverage regarding component interfaces, interactions, and control and data coupling is recommended. Concrete entity types of interfaces and interaction elements are not mentioned. From the perspective of complex system operations, the coverage of functions, function calls, and function sequences is demanded. This criterion is important for the coverage of complex use cases and distributed system operations.

ISO 26262 and DO-178C explicitly support the use of formal models, especially behavior models, for development and QA activities. Different notations are mentioned and recommended, for example transition-based notations (finite state machines), pre-/post-based notations (Z), and operational and concurrent notations (Petri nets). More information on the modeling notations and their classification is provided in [9].

The conclusions for functional integration testing on the model and software levels are: Requirements coverage is the major criterion and the most important goal of testing. Concrete techniques and criteria for creating requirements-based test cases are mentioned. The specification of complex and distributed system operations represents high-level requirements of the integrated system that have to be checked intensively. Additionally, the coverage of the software architecture as well as that of component interfaces and their interactions has to be considered for test design and test specification. Based on our experience from the MBAT project, no industrially proven automated test approach is available that sufficiently and efficiently covers standard-compliant functional integration testing on the model and software levels.

III. State of the Art

Model-based approaches provide a high degree of automation regarding analysis, transformation, and generation of valid execution sequences due to their sound mathematical basis. Different modeling notations have been systematically exploited for the generation of test cases. The set of corresponding techniques is called *model-based testing* (MBT). MBT approaches address those 40% of testing effort that are usually spent on test preparation and test specification in an industrial project [32]. By automating these activities, the overall testing effort can be significantly reduced. The standards analyzed in the previous section also propose the use of models for development and QA activities. The main focus of the MBT techniques is on functional testing, i.e., on testing against the functional specification [9]. Functional testing usually requires the specification of test cases with the corresponding pre-conditions, actions, expected results, and post-conditions.

Models that are constructed for the primary use of generating test cases from them are called *test models.* They represent the relevant information from the test and QA perspectives. Test models can describe intended and unintended functionality, unexpected and unspecified usage, or misuse to support robustness testing. Additionally, test models also guide and facilitate test case generation by providing information on importance and criticality or on the frequency of certain scenarios. An overview of test modeling notations and test case generation approaches is provided by Utting et al. in [9]. For model-based integration testing, numerous approaches have been developed with different kinds of test objectives and modeling notations. A detailed overview and a classification are given by Bauer and Eschbach in [31]. The approaches have been classified intro three classes (component-based, scenario-based, and combined approaches), which are described below. There are additional approaches that work directly on program code and code-based integration testing, but they are not considered here due to restricted code access in most industrial projects and missing support for testing high-level requirements.

The remaining solutions of the three classes have been assessed regarding their capabilities for modeling the two dimensions of integration testing: the low-level *interaction-focused view* and the *high-level requirements view,* which is related to *distributed system operations*. Due to the high complexity of the software systems and their requirements, a *stepwise assembly strategy* and the *composition of operations* should be supported. Test scenarios require the specification of their pre- and post-conditions. Therefore, the notation should also support the modeling of such *execution conditions*. Finally, the approach should be able to describe the *interaction patterns* regarding the system components and their interfaces as part of the operational implementation.

The *component-based integration test approaches* use dedicated behavior models, mostly different types of finite state machines, from the component perspective as the basis for test case generation. Their origin is the conformance and interoperability testing of protocols [10]. For integration testing, different finite state machine notations are used to represent the system behavior. Most of them focus on the coverage of synchronized events, e.g., the approaches by Koppol et al. [12] and Robinson-Mallett et al. [14]. Other approaches use extended finite state machines with variables and guards and define specific criteria for additionally covering data coupling and data flow dependencies on the subsystem level [13][15][16]. Component-based approaches usually support stepwise system assembly and integration testing. The main problem is that complex scenarios and high-level requirements are not sufficiently considered due to the focus on specific component interactions.

The *scenario-based approaches* focus on the modeling of high-level system requirements, system operations, use cases, and usage scenarios. Most of them use UML behavior diagrams such as sequence diagrams, collaboration diagrams, interaction or activity diagrams [18][19]. Each scenario (including rare and exceptional cases) has to be modeled explicitly. A second group applies operational modeling notations that consider concurrency like Petri nets [20] and Communicating Sequential Processes (CSP) [21]. The operational modeling notations have advantages in terms of model composition, but weaknesses regarding the description of operational execution conditions. Due to the high-level view focusing on usage scenarios, low-level aspects such as concrete component interfaces and component interactions are not covered sufficiently by all scenario-based approaches. The strategy of stepwise assembly is not considered by any of the approaches.

The most advanced solutions consider the heterogeneous aspects of functional integration testing: the high-level system features, operations, and requirements on the one hand and the concrete component interactions on the other hand. The approaches are classified as *combined integration test approaches.* They use different kinds of finite state machines to model the low-level behavior on the component and subsystem levels and a high-level model to describe the relevant usage scenarios and high-level requirements. For modeling the scenarios, different notations are used, such as finite state machines in the approach by Wieczorek et al. [24], UML collaboration diagrams in the solution by Ali et al. [22], or a tree-like feature interaction model in the publication by Benz [23]. All approaches support at least simple solutions for the composition, the description of the operational execution conditions, and the modeling of component interactions as operational implementations, but no approach exists that completely covers all aspects to the full extent. However, the approach by Benz [23] supports different kinds of composition operators and the one by Wieczorek et al. [24] supports the detailed modeling of component interactions.

The conclusion of the state-of-the-art analysis is that heterogeneous integration test approaches provide the most appropriate solutions for our problem. They are able to cover high-level system operations as well as low-level component interactions. None of the state-of-the-art approaches sufficiently supports all requirements stated. The composition of system operations, the modeling of complex execution conditions, and their implementation as component interplay is only partially solved by the available solutions. Therefore, we have developed a new model-based test approach that tackles these challenges.

IV. TEST MODELING NOTATION

The efficiency of MBT highly relies on the selection of an appropriate modeling notation and the availability of efficient model analysis technologies. The notation influences the quality and efficiency of model construction, i.e., the formalization of the requirements, and test case generation, i.e., the derivation of traces from the model.

The selection of an appropriate modeling notation depends on the characteristics of the system and its functionality to be modeled. As described above, the application type is the software level of embedded systems. In embedded systems, two types of functions are usually provided: computation and control functions. Computation functions are mainly used for connecting the system with its environment via sensors and actuators and for deriving relevant variables and decision points.

Control functions are connected to the system state and modes. They are usually used at a higher abstraction level than computation. Based on the stimulation pattern and the current system state, the future behavior is controlled.

MBT often focuses on testing the functional behavior and the control functionality. The functional behavior is expressed by stimuli, responses, pre- and post-conditions, and state variables. Especially for technical software systems, inputs and outputs can be complex due to time dependency and concurrency. Solutions have been proposed particularly for the Simulink/Stateflow simulation environment [26]. The following subsections will describe the generic heterogeneous test modeling approach (in subsection A), its instantiated modeling notations (in subsections B and C), and the concept for assuring consistency of the two notations (in subsection D).

A. *Towards a generic test modeling approach*

In order to enable fully automated test case generation, test models have to describe system-specific aspects, such as the system structure and component interfaces, as well as test-specific aspects, such as the importance of scenarios, interfaces, and modes. Most MBT approaches use one modeling notation as a basis for generating test cases [9]. In order to clearly divide the responsibilities of the model artifacts, the *generic heterogeneous test modeling approach* ER!S has been developed. The approach distinguishes between a low-level model that represents the test-relevant system behavior and a high-level model for representing complex requirements and guiding test case derivation. In the MBAT project, the approach has been instantiated for functional software integration testing.

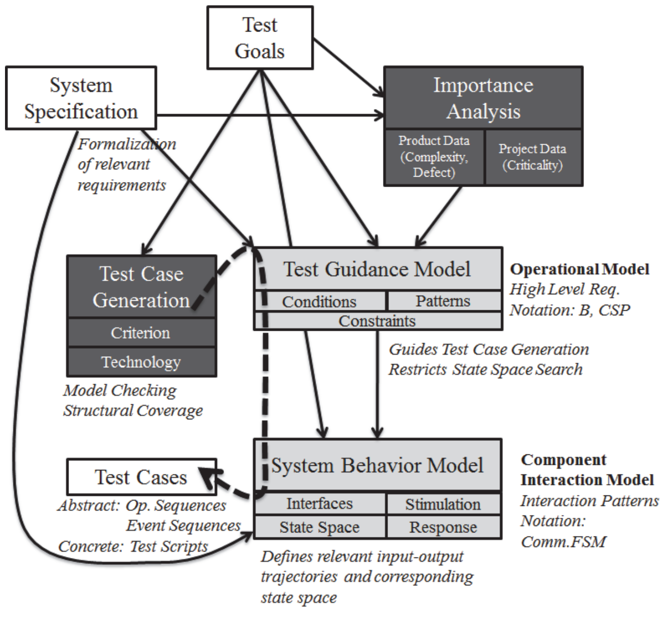

Figure 2. Generic test modeling approach

Figure 2 describes the artifacts involved in the generic test modeling approach and their relations. The starting point is the *system specification*, which is the initial source for describing the system functionality to be checked. In most cases, the system specification is a textual requirements document enriched with architectural descriptions of the components, interfaces, and communication middleware. The *test goals* describe the generic objectives of quality assurance, such as coverage of the functional requirements, assuring robustness in unspecified situations, or considering the most critical usage scenarios. Test goals influence the QA strategy and therefore also test modeling and test case generation in MBT.

Based on the specification and the test goals, an *importance analysis* is conducted, which considers the complexity and defect data of the product and other criticality factors of the test project. The importance analysis influences the abstraction level of the test modeling and the inclusion and exclusion of elements and requirements. Additionally, the importance analysis may serve as input for the distribution of the test effort, the selection of the test coverage criteria, and the guidance regarding test case generation.

In the approach, two types of test models are constructed: the *system behavior model* (SBM) and the *test guidance model* (TGM). The SBM defines the relevant system behavior for the test on an appropriate abstraction level. The SBM is constructed from the system specification and describes the interfaces, valid input-output trajectories, and the state space of the test object. For this work, the SBM focuses on component interactions. Therefore, the SBM for functional integration testing is also called *interaction test model* (ITM). Due to the characteristics of the actual test object of the evaluation, a discrete control system, the modeling notation for the ITM is a subset of timed automata [29]. This notation enables the description of a component-based system whose parts are synchronized by events. Communication protocols and specific middleware entities such as bus controllers can be modeled as additional state-based components of the SBM.

The TGM describes patterns and constraints for the application and exploration of the SBM and the conditions under which they are to be applied. Constraints are defined to prohibit or enforce defined situations for the forthcoming test case generation. The TGM can also represent the operational profile of the test object, which may differ in different environments. For this work, the TGM has to deal with composite system operations and functional scenarios with defined execution conditions. Therefore, the TGM for functional integration testing is called *operational test model* (OPM). A concrete operational implementation is defined by the interaction patterns of the system components. Due to the strong focus on the composition of different operations and the efficient description of their execution conditions, a heterogeneous notation based on B machines [27] and CSP [28] has been chosen. The integration of B machines and CSP for formal verification purposes has been shown in [30].

For the *test case generation* step, the coverage criteria and the generation technology have to be defined [9]. Considering the ER!S models, the coverage criteria determine the class of relevant elements of the test model that shall be covered by the test cases. Examples are the coverage of component interfaces or the coverage of conditional execution paths within an operation. The test cases are represented by sequences of operations and events, which are refined to executable test scripts (like C-Unit scripts [38] or signal descriptions in the Matlab / Simulink environment [37]).

B. *The Operational Test Model*

The OPM is used to guide the selection of test cases from an operational point of view. It describes the high-level functional requirements and system operations with their composite implementations and execution conditions, and the system state space. The implementation of the modeling notation is based on B machines and CSP. The OPM is defined as: *OPM = {Op,* Var_{Op}*,*S_{OP}*,* s_0*,* S_{Exit}*}*. It contains a finite set of hierarchical operations *Op*. The composition relation is defined by a partial ordering function, which enables operational composition with different operators. These operators are based on the composition operators of CSP [28]: sequencing, alternatives, conditional branching, and parallel interleaving. They enable the construction of so-called *composite operations*. The non-composite operations are called *basic operations*. A finite set of variables Var_{OP} is defined to model the system states and operational execution conditions. Combinations of variable values define the system state space. A dedicated start state s_0 and an optional set of ending states S_{EXIT} for the execution of test case are defined. For the OPM, two graphical representations have been developed to facilitate discussions and model reviews: the Operational Hierarchy Graph (OHG) and the Operational Composition Graph (OCG).

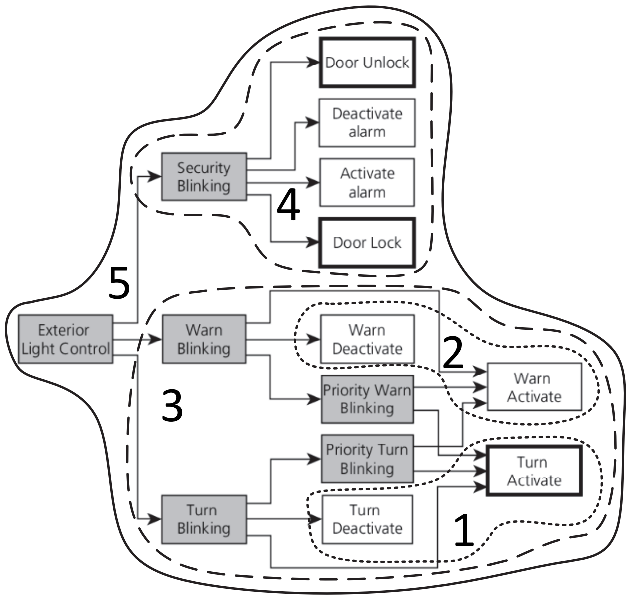

Figure 3. Operational Hierarchy Graph for the sample application

Figure 3 shows an OHG, which represents the hierarchy of operations regarding their composition relations. Boxes represent operations. Basic operations are marked white, composite operations are marked gray. The arrow points to the sub-operations of a composite operation. The example shown is taken from the evaluation case study described in chapter VI. Additionally, a valid integration order for the system operations is annotated, which consists of five steps. The integration is performed from lower-level to higher-level operations, i.e., from step #1 to step #5.

The OCG visualizes the operational composition with a directed graph. An example of an OCG is shown in Figure 4. It describes the composite operation *Warn Priority Blinking* of the example used in the feasibility example. The rectangular boxes represent the sub-operations referenced and rounded rectangles represent the execution conditions (here: pre- and post-condition) with the corresponding Boolean formulas. Composition operators are shown with specific symbols, such as arrows for sequencing and diamonds for alternatives. The OCG traversing starts in the pre-condition node and ends in one of the post-condition nodes. Every trace through an OCG is a valid operational execution.

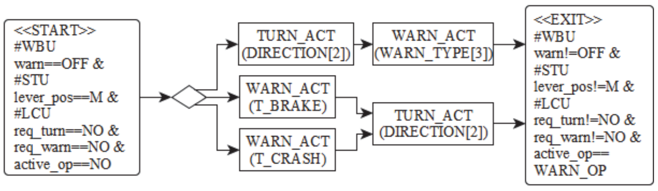

Figure 4. Operational Composition Graph of an operation

The operation of the example deals with the determination of the active blink operations when multiple turn and warn blink operations (manual, emergency brake, and crash) are requested. The interesting cases in the example are when (1) a previously activated turn blink operation is overwritten by a subsequent warn blinking (manual, emergency brake, and crash) and (2) a previously activated emergency brake and crash warn blinking is deactivated by subsequent turn blinking.

C. *The Interaction Test Model*

The ITM describes concrete interactions between system components in order to implement an operation. Its notation is a subset of timed automata [29]. The ITM is defined as the parallel composition of a set of component models (*CM*) that may synchronize on shared events. A CM is defined as $CM = (L, l_0, Var_{Comp}, Act, E$. Locations (*L*) represent the vertices of the component automaton connected by a set of edges *(E)*. Every CM has a designated initial location (l_0) and a set of variables (Var_{comp}), which is a subset of the operational system variables (*Var*). Furthermore, a CM has an alphabet of events with inputs and outputs (*Act*). Edges (E) connect two locations. They are annotated with the corresponding input (?) or output (!) event. The operations are implemented by a set of component interactions, which are related to concrete component edges. Therefore, the ITM component edges are annotated with the set of operations that are connected to them.

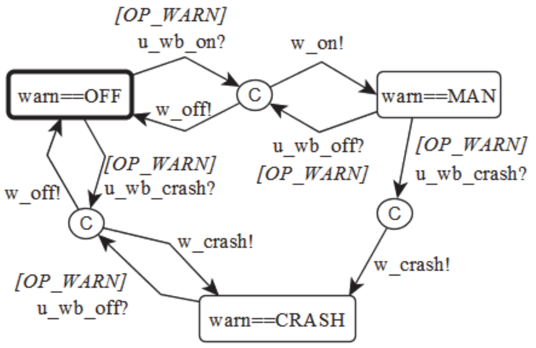

Figure 5. Sample interaction test model of a component

Figure 5 shows a sample ITM for the component *WarnBlinkUnit* of the feasibility study. The graphical representation is similar to common finite state machine notations. The ITM locations are expressed as rounded nodes (stable states) or circles (committed states); the transitions are represented as directed edges. The stable component states are annotated with an invariant, which is a unique combination of component state variable values. In the example, only one variable is used. A state is stable if all of its outward transitions are only enabled by external stimulation [33].

Transitions are labeled with at most one event, which is either a sending event (!) or a receiving event (?). The relevance of transitions for the implementation of certain operations is annotated by guard conditions. In our example, the model transitions are used for the warn blink operations, which are expressed by the Boolean variable *OP_WARN*. For analysis and test case generation, the transition guards help to reduce the complexity of the artifacts and focus on the relevant parts.

The component behavior is defined by a sequence of one input and a list of outputs, which have stable source and target states. Since timed automata allow only one event per transition, the input and the corresponding output(s) are constructed as an atomic sequence of transitions connected by committed states. During the exploration of the system state space, committed states have to be left immediately by taking an outgoing transition when they are traversed. This assures the atomicity of the event sequence.

D. Model Construction and Analysis

Complex formal models that are created manually from potentially incomplete and inconsistent sources require a systematic construction process and intensive QA. Another issue is the use of different model types, which may produce consistency issues. The construction approach provides a systematic procedure for designing the test model and applies guidelines and restrictions to reduce the fault-proneness and complexity of the artifacts. A formal correctness proof of a complex model is difficult to achieve. Therefore, a stepwise heuristic procedure is applied, comprising parallel construction and model analysis activities. The approach is supported by prototypical tools. For the conduction of the model analysis, the external tools Uppaal [35] and ProB [36] were used.

Both models, OPM and ITM, were checked independently regarding certain properties such as deadlocks and reachability of elements. Further analysis activities assured the consistency of both models. A catalogue of concrete analysis activities was defined, which is described in part below.

As shown in Figure 2, the main source for the test modeling is the system specification, which contains all information about the static system structure of the test object and its functionality and operations. The construction approach of ER!S models was derived from sequence-based specification (SBS, [34]) which enables the systematic specification of component test models. The system functionality in ER!S is specified as a set of operations that are implemented as interactions between components under defined conditions.

The recommended construction approach from the operational view is bottom-up. According to the operational hierarchy, basic operations are specified first with their execution conditions and interaction patterns. These interaction patterns describe event flows, sequences of inputs and corresponding outputs, and conditions under which they are applicable. The system is supposed to run in a so-called slow environment [33]. This means that the system is only stimulated when all of its components are in stable states, i.e., the components do not perform autonomous interactions. All component responses are direct reactions to stimulation from the environment. Operations always start and end in stable system states, which facilitates the construction of deterministic test models. This leads to special requirements for the event sequences and states that are checked in the ITM analysis. Furthermore, the ITM is checked for interoperability, i.e., the ability of communicating via its interfaces.

In the next step, the composite operations with their composition patterns and the execution conditions are specified. In the subsequent analysis, the OPM is checked for the validity and executability of the composition patterns. The OPM analysis checks whether operational traces exist that completely traverse the operational specification.

The quality of the source documents affects the construction paradigm of the ER!S models. Faulty, inconsistent, incomplete, or even changing requirements lead to model design flaws and model changes. In order to assure compatibility and consistency between different modeling notations, two concepts are introduced that focus on the relations between operations and interaction. The first concept is an *injective mapping function* for OPM states and ITM states. Each state of the OPM state space is mapped to a unique stable system state of the ITM. The reachability of selected stable states of the ITM is checked. Specific requirements for stable states regarding variable values and transition events are defined and checked as well. The second concept are *operational tags*, which are annotated to ITM component transitions. For each operation, the corresponding sub-model of the ITM is determined. The ITM analysis assures that the interaction patterns of the operations are executable and valid regarding the conditions and variable values.

V. Generation of Test Artifacts

After the construction of the test model and its verification, test cases are derived as ER!S model traces. The test case set comprises valid model traces that cover selected test requirements. For the generation of test cases, many results of the ER!S model analysis can be reused since they contain sample model traces that prove the reachability of defined model elements.

Figure 6 shows the generation of test artifacts from the test models. The starting point are the TGM and the SBM, which are analyzed in order to define the *assembly strategy* for the system components and operations. An operation-driven bottom-up strategy is proposed, i.e., operations of a lower hierarchy level are integrated earlier, whereas complex operation of higher levels are integrated later in the test stage [31]. An example of a valid assembly strategy for a feasibility study is annotated in Figure 3. Each step covers a disjoint set of the selected test requirements. Two criteria have been developed that enable the scalable assembly of components and operations. The first criterion determines the relevant sub-systems that perform specific operations. The other criterion determines the integration strategy for the relevant system operations in each integration step. Since each assembly step focuses only on specific aspects of the system functionality, only a subset of the test models is required for test case generation. Therefore, step-specific reduced test models are created that only contain the required subset of the information.

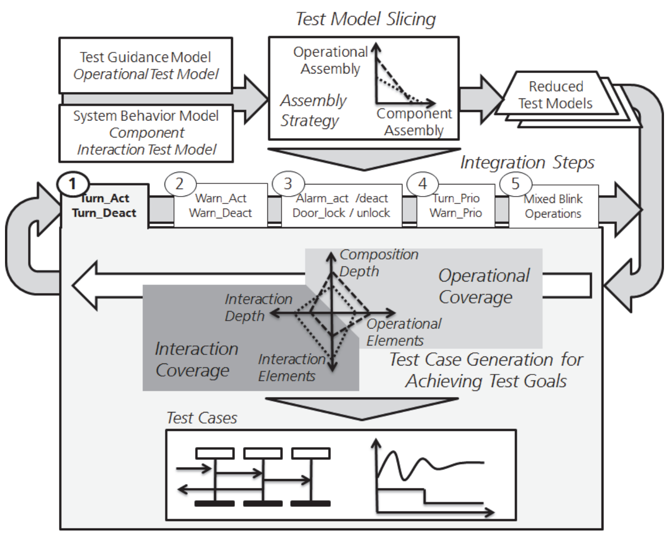

Figure 6. Workflow for generating test artifacts

System operations have different kinds of test-relevant information depending on their implementation and composition. Basic operations focus on the coverage of interaction patterns under specific conditions. Composite operations focus on the coverage of their alternative composition paths under specific conditions. Therefore, our test approach provides a set of *coverage criteria* for component interactions and for operational composition. The interaction coverage criteria are related to component interfaces, event-sending and receiving transitions, and synchronizations of them. The operational coverage criteria consider the different execution conditions and the characteristics of the composition operators used in the implementation.

The test case generation procedures use the model checking capabilities of the tools Uppaal and ProB, which are capable of verifying properties and deriving sample traces for timed automata, B machines, and CSP models. The ER!S tool transforms the coverage criteria and sets of test requirements into simple model checking queries for the tools mentioned above. The resulting test cases are valid ER!S model traces. They consist of sequences of operations implemented by event traces. More details on the test selection criteria and generation technologies are provided by Bauer and Eschbach in [31].

VI. EVALUATION

The evaluation was conducted in the MBAT project together with tool vendors and product manufacturers from the transportation domains. The goal was to assess the impact of the new heterogeneous MBT technique on the test process compared to manual expert-driven requirements-based test case creation and a simple MBT technique with finite state machines, which had been already introduced to the companies. The properties to be evaluated were: (A) compliance with the recommendations of the process standards, (B) coverage of the test cases regarding the properties to be checked, and (C) the manual effort spent on the construction of the test artifacts.

The evaluation was planned to be conducted in two rounds: (1) a feasibility study to initially assess the new test approach and (2) a detailed quantitative evaluation study to measure the impact. In the following subsection, the test object, the results of the first evaluation round, and the set-up for the second round are presented.

There were several challenges that complicated an evaluation in the MBAT project. The first challenge was the missing independence of system experts and test experts (for the expert-driven requirements-based test case derivation), which might have influenced the significance of the results. The other issue was the confidentiality of the test object in the project, which restricted the usage and publication of certain details. Therefore, a new test object with the corresponding specification documents, design models, and executables was created. The functionality is close to the features of the actual test object, but the system structure, component interfaces, and the concrete implementations were simplified and developed independently to abstract from any confidential details.

A. The test object

The test object of the evaluation case study was a simplified version of an executable design model (notation: Simulink / Stateflow) of an automotive exterior light control system (ELCS). The ELCS consists of five system components: steering unit, ignition unit, warn blink unit, door control unit, and exterior light control unit. Another eight environmental components were considered for the evaluation of stimulation and response. The functionality comprises several blinking functionalities of the external lighting, including turn indication, warn blinking, and security features such as door locking and theft alarm.

TABLE I. PRIORITIES OF BLINK OPERATIONS IN THE CASE STUDY

Prio	**Class**	**Blink operation**	**Duration**	**Side**
1	Warning	Crash warning	Permanent	Both
2		Manual warning*	Permanent	Both
3		Brake warning	Permanent	Both
4	Turn indication	Permanent*	Permanent	Left/right
5		Temporary*	Temporary	Left/right
6	Security	Theft alarm	Permanent	Both
7		Door locking	Temporary	Both
8		Door unlocking	Temporary	Both

Table I shows the different blink functionalities with their priorities (1 – highest, 8 – lowest) and the properties that were checked in the test evaluation. Several blink operations can be requested at the same time, but only one operation can be active. If multiple blink operations are requested, the operation with the highest priority is selected and executed. The only priority exception in the example application is that turn indication overwrites an active warn blinking in certain situations (marked with * in the table).

Each functionality has defined pre-conditions for its activation and deactivation, for example regarding the ignition status or door locking status. The observable outputs of each functionality are the flashing side markers outside the car and the flashing LEDs on the car's dashboard at defined frequencies. The key issue of functional integration testing is to assure the correct execution of the blinking behavior in the case of multiple activated blinking functions.

B. Preparation of the evaluation

As the first step of the evaluation, a simplified requirements specification of the ELCS was developed based on the knowledge of system and test experts, the original requirements, and the existing test goals and test cases. The new specification does not contain confidential information and abstracts from irrelevant details for functional integration testing. Based on the simplified specification, the executable test object, i.e., the application under test, was developed as an executable Matlab / Simulink model [37], which enables the automated generation of program code. In the specification and design activities, experts with system and domain knowledge were involved as well as test experts.

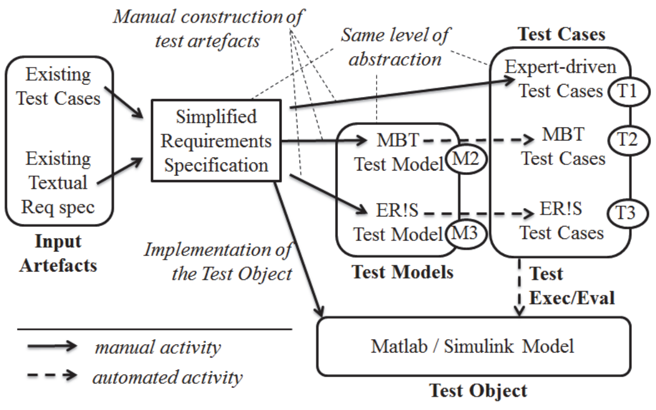

Figure 7. Evaluation set-up

Figure 7 shows the set-up of the evaluation, which comprises the development of the actual test object, the construction of the test artifacts, and the derivation of the test cases for the following three methods: (1) manual, expert-driven, and requirements-based test case derivation, (2) MBT with finite state machines, and (3) MBT with heterogeneous test models (the ER!S approach).

After the creation of the test object, the test artifacts for the different approaches were created based on the same simplified requirements specification. The manual creation of the expert-driven test cases (T1) was done by system experts according a standard-compliant, expert-driven, and requirements-based approach. The construction of the test models for the MBT with finite state machines (M2) and the MBT with heterogeneous models (M3) was done by dedicated method experts. For the construction of the test artifacts (models and test cases), the same abstraction level regarding system structure, component interfaces, events, and variables was applied. The test cases for both MBT approaches were created automatically, which is displayed as dashed lines in the figure.

C. Results from the feasibility study

In the first evaluation round, we aimed at a short assessment of the test technology. Therefore, we considered a subset of the systems' functionality. Test models and test cases were created by applying each of the three methods. The assessment regarding the selected properties is summarized in Table II. The complexity of the different test artifacts could not be assessed adequately since the test approaches use different modeling paradigms (test sequences, finite state machines, and the heterogeneous notation based on timed automata, B, and CSP).

TABLE II. SUMMARY OF FIRST EVALUATION RESULTS

	Expert/req-based (T1)	**MBT-FSM (M2, T2)**	**ER!S (M3, T3)**
Standard Compl.	+	-	+
Operational Test Coverage	81%	67%	90%
Interaction Test Coverage	92%	100%	100%
Test Effort	100%	119%	135%

The first evaluation aspect is standard compliance, which qualitatively assesses the considerations of the recommendations from the process standards mentioned in section II. The ER!S approach was developed with the intent of being compliant with the industrial standards and guidelines. The support of certain topics, such as coverage of functional requirements, interactions, and operational sequences, is sufficient and comparable to the expert-driven test approach that has been applied to the test project and the resulting certification for many years. The MBT approach with finite state machines does not sufficiently consider the characteristics of more complex system operations.

The next aspect is test coverage, which is a quantitative quality criterion of a set of test cases regarding a set of properties and test requirements. It can be seen as an indicator of the quality of the test process. The ER!S approach facilitates the determination of appropriate criteria regarding the component interactions and the operational implementation. The initial evaluation of interaction and operational coverage showed that ER!S test cases (T3) achieved high coverage of both criteria (100% regarding the interactions and 90% regarding the operational aspects). The MBT test cases (T2) achieved full coverage of the selected interactions, which is explained by the strong focus on component behavior and communication. Therefore, the operational coverage of T2 is also much lower (67%) than with the other approaches. The expert-based test cases (T1) had reasonably high coverage of both criteria (92% regarding the interactions and 81% regarding the operational aspects). An influencing factor for the detailed assessment is the varying degree of importance of selected test requirements, which was not considered in the initial evaluation. The discussion with the industrial partners showed that the automated test approaches with their test case sets T2 and T3 contained a slightly higher number of less relevant elements. In the next evaluation round, a more detailed analysis of the test coverage will be conducted.

The reduction of effort and costs is an important success factor when it comes to introducing new technologies. In our feasibility study, the effort for manually constructing the test artifacts was assessed (T1, M2, M3). The main part of the effort for all approaches was spent on determining and defining the components, interfaces, events, variables, and interaction patterns in order to ensure the same origin and abstraction level of the test cases. Automated steps, such as the test case generation for T2 and T3, were not considered. The initial effort for constructing the first set of test models and test cases (+19% for M2 and +35% for M3) is slightly higher for the model-based approaches than for the traditional testing approach (T1). This is caused by the fact that T1 only contains selected scenarios for the application. Test models and the resulting test case sets are often more complete regarding the selected properties. A significant effort reduction for the model-based approaches is expected when existing test artifacts are incrementally extended for updated product versions and similar systems.

The limitations of the feasibility study only enable an initial and rough assessment of the impact and capabilities of our ER!S approach. The standard compliance, the higher test coverage, and the slightly higher test effort in the first round are indications that ER!S is an efficient and reasonable integration testing approach. Further evaluations are needed to assess the capabilities and impact on the overall test processes in detail.

VII. Conclusion and Future Work

In this article, we presented the novel model-based approach ER!S for functional integration testing of software-intensive technical systems. The detailed analysis of the recommendations and challenges stated in the two relevant process standards (ISO 26262 and DO-178C) resulted in a set of major requirements for functional software integration testing that can be addressed by model-based solutions. The state-of-the-art approaches do not sufficiently cover the multifaceted aspects of integration testing with the two dimensions of composite and distributed system operations and the actual component interplay that implements these operations.

The ER!S approach is able to efficiently tackle these challenges. It comprises a heterogeneous modeling notation that considers both aspects of functional integration testing. The notation enables the automated generation of test cases using different coverage criteria. The results of the first evaluation round were positive. Our approach produced test artifacts of higher test quality regarding the test coverage. More detailed results will be gathered in the second evaluation round. Other future activities will comprise the improvement of the analysis and test case generation algorithms and the extension of the tool chain, which currently consists of a set of loosely coupled in-house, external research, and commercial tools.

Acknowledgment

This work has been funded by the ARTEMIS project “MBAT” (grant number: 269335). We would also like to thank Sonnhild Namingha for proofreading.

References

[1] Website of the project MBAT,,http//www.mbat-artemis.eu [12 Aug 2014]

[2] P. Liggesmeyer and M. Trapp, “Trends in Embedded Software Engineering”, IEEE Software 26(3), pp. 19-25, Jan. 2009.

[3] M. Maurer and H. Winner, Automotive Systems Engineering, Springer June 2013

[4] J. Zander, I. Schieferdecker, P. Mosterman, Model-Based Testing For Embedded Systems, CRC Press, Sep. 2011

[5] IEC 61508, International Electrotechnical Commission, IEC 61508:2010 - Functional safety of electrical/electronic/programmable electronic safety related systems, 2010

[6] ISO 26262, International Standardization Organization, ISO 26262:2011 - Road vehicles – Functional safety, 2011

[7] DO-178C,.Radio Technical Commission for Aeronautics Software, DO-178C:2011 Considerations in Airborne Systems and Equipment Certification, 2011

[8] EN 50128, CENELEC - European Committee for Electrotechnical Standardization, EN 50128:2011, Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, 2011

[9] M. Utting, A. Pretschner, B. Legeard, “A Taxonomy of Model-based Testing Approaches”, Softw. Test. Verif. Reliab. 22(5), pp. 297-312, Aug. 2012

[10] A. V. Aho, A. T. Dahbura, D. Lee, M. Uyar, “An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours”, IEEE Transactions on Communications 39(11), 1604-1615, Nov. 1991

[11] Y. Wu, D. Pan, M.-H. Chen, “Techniques for Testing Component-Based Software”, Proceedings of the 7th Int. Conf. on Engineering of Complex Computer Systems, IEEE Computer Society, pp. 222-232, June 2001

[12] P. V. Koppol, R. H. Carver, K.-C. Tai, “Incremental Integration Testing of Concurrent Programs”, IEEE Trans. Software Eng. 28(6), pp. 607-623, June 2002

[13] A. Desmoulin and C. Viho, “A New Method for Interoperability Test Generation”, Proceeding of the TestCom/FATES'07, Springer, pp. 58-73, June 2007

[14] C. Robinson-Mallett, R. Hierons, J. Poore, P. Liggesmeyer, “Using Communication Coverage Criteria and Partial Model Generation to Assist Software Integration Testing”, Software Quality Control 16(2), pp. 185-211, Apr. 2008.

[15] L. Gallagher, J. Offutt, A. Cincotta, “Integration testing of object-oriented components using finite state machines”, Softw. Test., Verif. Reliab. 16(4), pp. 215-266, Jan. 2006

[16] F. Saglietti, N. Oster, F. Pinte, “Interface Coverage Criteria Supporting Model-Based Integration Testing”, ARCS '07 - Workshop Proceedings, VDE Verlag GmbH , pp. 85—93, 2007

[17] A. Pretschner, O. Slotosch, E. Aiglstorfer, S. Kriebel, “Model-based Testing for Real: The Inhouse Card Case Study”, Int. J. Softw. Tools Technol. Transf. 5(2), 140—157, 2004.

[18] A. Abdurazik and J. Offutt, “Using UML Collaboration Diagrams for Static Checking and Test Generation”, Proceedings of the 3rd International Conference on The Unified Modeling Language: Advancing the Standard, Springer-Verlag, pp. 383—395, 2000

[19] F. Basanieri and A. Bertolino, “A Practical Approach to UML-based Derivation of Integration Tests”, in 4th International Software Quality Week Europe, Nov. 2000

[20] H. Reza and E. Grant, “A Method to Test Concurrent Systems Using Architectural Specification”, J. Supercomputing 39(3), pp. 347-357, Feb. 2007

[21] S. Nogueira, A. Sampaio, A. Mota, “Guided Test Generation from CSP Models”, Proceedings of the 5th International Colloquium on Theoretical Aspects of Computing, Springer, pp. 258-273, Sep. 2008

[22] S. Ali et al., “A State-based Approach to Integration Testing Based on UML Models”, Inf. Softw. Technol. 49(11-12), pp. 1087-1106, Nov. 2007

[23] S. Benz, “Combining Test Case Generation for Component and Integration Testing”, Proceedings of the 3rd International Workshop on Advances in Model-based Testing, ACM, pp. 23—33, May 2007

[24] S. Wieczorek et al., “Applying Model Checking to Generate Model-Based Integration Tests from Choreography Models”, 'Testing of Software and Communication Systems', Springer, pp. 179-194, Nov. 2009

[25] M. Utting, A. Pretschner, B. Legeard, “A Taxonomy of Model-based Testing Approaches”, Softw. Test. Verif. Reliab. 22(5), pp. 297-312, Aug. 2012

[26] F. Böhr, Model-based Statistical Testing of Embedded Real-time Software with Continuous and Discrete Signals in a Concurrent Environment: The Usage Net Approach, PhD thesis, TU Kaiserslautern, Dr. Hut, 2012

[27] J.-R. Abrial, The B-book: Assigning Programs to Meanings, Cambridge University Press, 1996

[28] C. A. R. Hoare, Communicating Sequential Processes, Prentice-Hall, Apr. 1985

[29] R. Alur, and D. L. Dill, “A Theory of Timed Automata”, Theoretical Computer Science 126 (2), pp. 183-235, Apr. 1994.

[30] M. J. Butler and M. Leuschel, “Combining CSP and B for Specification and Property Verification”, Proceedings of the 2005 international conference on Formal Methods, Springer, pp. 221-236, July 2005.

[31] T. Bauer and R. Eschbach, “Model-Based Testing of Distributed Functions”, Advanced Automated Software Testing: Frameworks for Refined Practice, CRC Press, pp. 151-181, Jan. 2012

[32] M. Pol, Software Testing: A guide to the TMap Approach, Addison-Wesley Professional, Nov. 2001

[33] M. Shahbaz, Reverse Engineering Enhanced State Models of Black Box Components to support Integration Testing, PhD thesis, Grenoble Institute of Technology, 2008

[34] S. J. Prowell, and J. H. Poore, “Foundations of Sequence-Based Software Specification”, IEEE Trans. Software Eng. 29(5), pp. 417-429, May 2003

[35] Website of Uppaal, http://www.uppaal.org/ [12 Aug 2014]

[36] Website of ProB, http://www.stups.uni-duesseldorf.de/ProB [12 Aug 2014]

[37] Website of Simulink http://www.mathworks.de//simulink [12 Aug 2014]

[38] Website of C-Unit http://cunit.sourceforge.net/ [12 Aug 2014]

APPENDIX

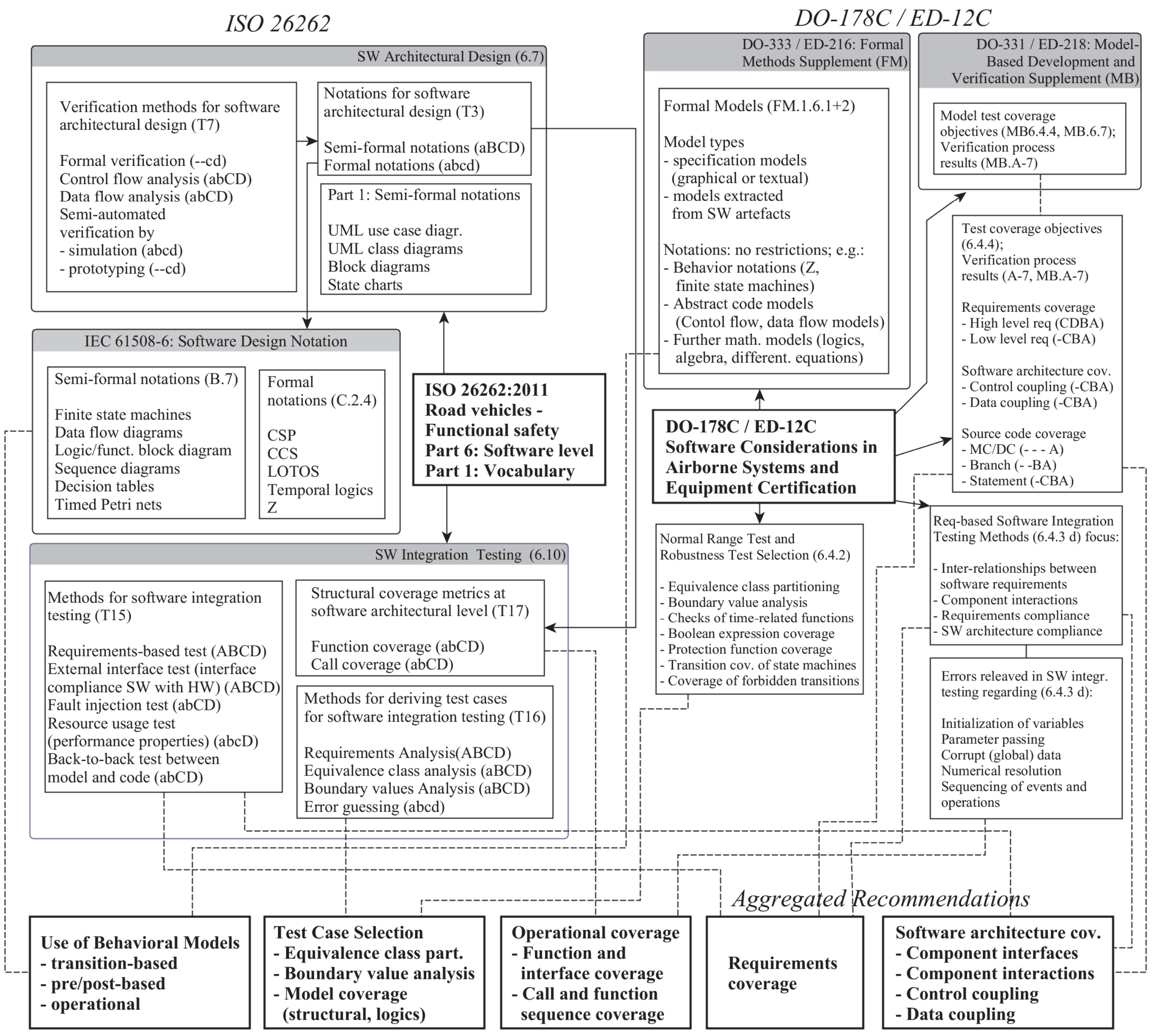
ISO 26262
DO-178C / ED-12C
SW Architectural Design (6.7)
Verification methods for software architectural design (T7)
Formal verification (--cd)
Control flow analysis (abCD)
Data flow analysis (abCD)
Semi-automated verification by
- simulation (abcd)
- prototyping (--cd)
Notations for software architectural design (T3)
Semi-formal notations (aBCD)
Formal notations (abcd)
Part 1: Semi-formal notations
UML use case diagr.
UML class diagrams
Block diagrams
State charts
IEC 61508-6: Software Design Notation
Semi-formal notations (B.7)
Finite state machines
Data flow diagrams
Logic/funct. block diagram
Sequence diagrams
Decision tables
Timed Petri nets
Formal notations (C.2.4)
CSP
CCS
LOTOS
Temporal logics
Z
ISO 26262:2011
Road vehicles -
Functional safety
Part 6: Software level
Part 1: Vocabulary
SW Integration Testing (6.10)
Methods for software integration testing (T15)
Requirements-based test (ABCD)
External interface test (interface compliance SW with HW) (ABCD)
Fault injection test (abCD)
Resource usage test (performance properties) (abcD)
Back-to-back test between model and code (abCD)
Structural coverage metrics at software architectural level (T17)
Function coverage (abCD)
Call coverage (abCD)
Methods for deriving test cases for software integration testing (T16)
Requirements Analysis(ABCD)
Equivalence class analysis (aBCD)
Boundary values Analysis (aBCD)
Error guessing (abcd)
DO-333 / ED-216: Formal Methods Supplement (FM)
Formal Models (FM.1.6.1+2)
Model types
- specification models (graphical or textual)
- models extracted from SW artefacts
Notations: no restrictions; e.g.:
- Behavior notations (Z, finite state machines)
- Abstract code models (Contol flow, data flow models)
- Further math. models (logics, algebra, different. equations)
DO-331 / ED-218: Model-Based Development and Verification Supplement (MB)
Model test coverage objectives (MB6.4.4, MB.6.7); Verification process results (MB.A-7)
Test coverage objectives (6.4.4);
Verification process results (A-7, MB.A-7)
Requirements coverage
- High level req (CDBA)
- Low level req (-CBA)
Software architecture cov.
- Control coupling (-CBA)
- Data coupling (-CBA)
Source code coverage
- MC/DC (- - - A)
- Branch (- -BA)
- Statement (-CBA)
DO-178C / ED-12C
Software Considerations in Airborne Systems and Equipment Certification
Normal Range Test and Robustness Test Selection (6.4.2)
- Equivalence class partitioning
- Boundary value analysis
- Checks of time-related functions
- Boolean expression coverage
- Protection function coverage
- Transition cov. of state machines
- Coverage of forbidden transitions
Req-based Software Integration Testing Methods (6.4.3 d) focus:
- Inter-relationships between software requirements
- Component interactions
- Requirements compliance
- SW architecture compliance
Errors releaved in SW integr. testing regarding (6.4.3 d):
Initialization of variables
Parameter passing
Corrupt (global) data
Numerical resolution
Sequencing of events and operations
Aggregated Recommendations
Use of Behavioral Models
- transition-based
- pre/post-based
- operational
Test Case Selection
- Equivalence class part.
- Boundary value analysis
- Model coverage (structural, logics)
Operational coverage
- Function and interface coverage
- Call and function sequence coverage
Requirements coverage
Software architecture cov.
- Component interfaces
- Component interactions
- Control coupling
- Data coupling

Figure 8. Aggregated recommendations of the ISO 26262 and DO-178C regarding software integration testing