Model-based Testing and Verification of Dependable Systems

Armin Beer
External Contributor of Siemens AG Österreich / A-2500 Baden bei Wien
armin.beer.ext@siemens.com

Abstract

Testing of dependable event-based systems is very important to ensure that all requirements (including non-functional requirements such as reliability, availability, safety and security) are met, and the relevant standards are observed. In this paper we provide an overview of the state of the practice in testing dependable event-based systems and identify the challenges that have to be addressed in the future. We illustrate our findings by a case study for a transportation system. The most important topics for research and improvement are: (1) formal modeling techniques for domain experts, (2) smart monkey testing techniques for reliability testing (3) mutation analysis to reveal the defect-detection potential of test suites, and a (4) recommending tool for the selection of test-case design methods.

Keywords

Model-based testing, test-case design, test automation, randomized testing, mutation analysis

1. Introduction

Dependability is defined as: "the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers". Dependability includes the following attributes [1] availability, reliability, safety and security.

The following is typical for all these areas:

- Growing software/hardware complexity: A railway system, for example, consists of infrastructure, logistics and interlocking blocks: these require a diverse range of different hardware and software components to interact.
- Third-party integration and code reuse: Standard software, such as GUI builders, is integrated.
- Certification of software: Operating systems, test tools, etc. have to be certified. This often requires an extra budget.

Testing is a very important aspect of building dependable systems. Experience shows that systematic and thorough testing helps to produce high-quality systems and makes accurate quality statements possible. There is consequently a huge demand for new test-case generation approaches and tools.

Standards such as EN 50128 and IEC 61508 recommend the use of formal methods at higher safety integrity levels, but are not frequently used, because:

- The notations for the formal specification of systems are highly abstract (e.g. Z or VDM), making it hard for domain experts to apply those methods efficiently.
- The notion of defects and their effects is seldom considered systematically in testing.
- The available method guides do not encourage the use of systematic test-case design methods.

This article has five main sections. Firstly, we shall examine the question of testing safety-critical systems. The next section briefly outlines state-of-the-art testing of such systems. This will be followed by a case study about testing a railway system from Siemens Transportation Systems. A short discussion of the results and the lessons learned will also be given. The final section before the conclusion will present some answers to research questions and offer proposals for improvements in testing safety-critical systems.

2. State-of-the-art testing of dependable systems

The standards relevant for these kinds of projects are EN 50128 (software-based components), EN 50129 (railway signaling systems) and IEC 61508 (safety integrity level SIL). These standards recommend that testing should concentrate on robustness, compliance and compatibility. Requirements-based testing is mandatory. The DO-178B, DO-278 and DO-248 standards apply in the aerospace industry [2]. From the viewpoint of testing, the different DALs (Development Assurance Levels, from level A (catastrophic) to E (no safety effect)) define the required degree of testing. Level C, for instance, requires 100% verification of code-branch coverage [3] [4]. In Figure 1 we present a traceability matrix [5] for comparing the verification and test requirements for the standards IEC61508 and DO-178B. We focus on the verification and validation parts relevant to SIL (Safety Integrity Level) 4 or DAL A. The problem is that the standards do not describe how the recommended (R, HR) or mandatory (M) test techniques could be applied. The testing standards established by the ISTQB (International Software Testing Qualifications Board) bridge this gap between the

standards and their application in real world projects, at least in some parts. [6].

Verification group	Technique	IEC61508/ SIL4	DO-178B
Module test and integration	Dynamic analysis and test Functional and black-box test	HR HR	X X
Software safety validation	Functional and black-box test Performance tests Probabilistic tests ...	HR M M	X
Dynamic analysis and test	Error seeding Structural tests ...	R HR	X
Functional and black-box test	Boundary value analysis Equivalence class test ...	HR HR	X X
Modeling	Data flow diagrams State transition diagrams ...	HR HR	X

Figure 1: Traceability matrix

Questions of how to build up confidence or whether or not to apply a specific testing technique are considered in [7]. Background information on typically occurring bugs has to be taken into account in the test-case design. [8] A mature organization with processes at CMMI level 3 or 4 can develop systems of high quality. In order to reduce testing costs while keeping the quality level, improvements have to be made in test management, testability of requirements and test-case design, verification and validation. Even for a life-critical system, exhaustive testing is not feasible due to the combinatory explosion of test cases. Therefore, the right test cases need to be identified and developed on the basis of test-case design methods [9]. As shown in Figure 2, the ISTQB classification is oriented to the black-box and white-box testing strategies. However, we propose making a distinction between methods for the generation of test sequences and test data. Automated test-case generation can help to bridge the gap between specification documents and testing [10] [11]. However, formal models have to be used, for instance in UML.

ISTQB Classification →

Goal-oriented Classification ↓

	Black Box	White Box
Data-Oriented Methods	Equivalence Classes - Boundary Values Cause-Effect Graphs Classification Trees	Condition Testing
Sequence-Oriented Methods	State-Transition Testing: - Node Coverage - Transition Coverage - Path Coverage Use Case Testing	Control Flow Testing: - Statement Coverage (C0) - Branch Coverage (C1) - Path Coverage (C2)

Figure 2: Test-case design taxonomy

We will demonstrate the practical application of a state-of-the-art test pattern in a case study in the domain of transportation systems.

The case study is structured into the following paragraphs:

- A short description of the project.
- The testing approach – a description about the test process and the impact the standards had on testing.
- Results and lessons learned.

3. Case study: Test of an Interlocking System for Transportation Systems (TS)

The development and testing of these kinds of projects are regulated in the CENELEC standards such as EN 50128 (software for railway control), EN 50129 (railway signaling systems) and EN 61508 (safety of electronic systems). These standards promote the V-model for development and testing (Figure 3).

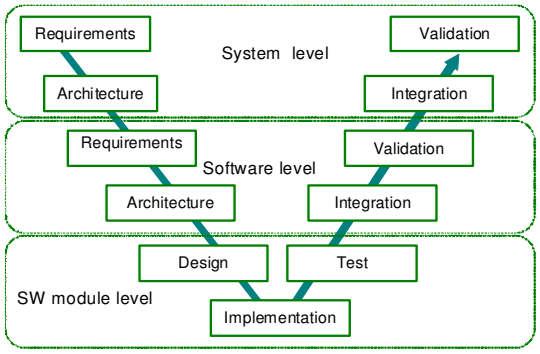


Figure 3: V-model applied to TS

3.1 The project

The railway case study looks at testing an interlocking subsystem software for railway control systems that consist of three layers: infrastructure, logistics and interlocking. The infrastructure represents a railway yard consisting basically of a collection of linked railway tracks equipped with features such as signals, points and level crossings.

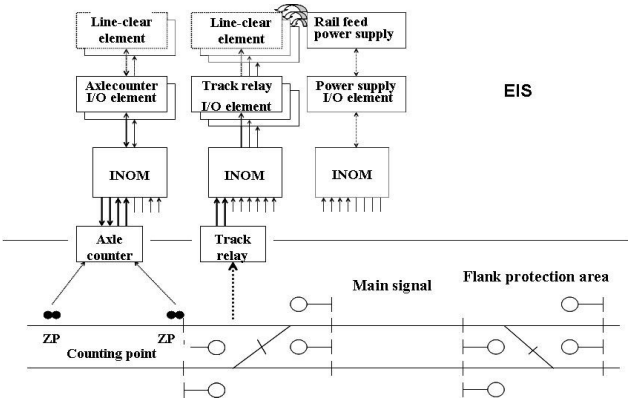


Figure 4: Electronic interlocking system (EIS)

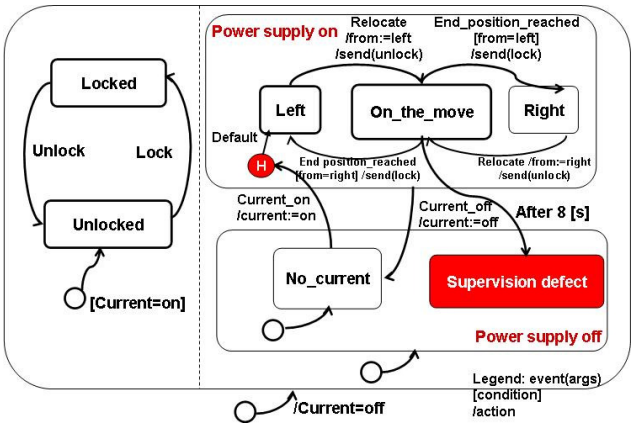
Since the interlocking subsystem is the most safety-critical layer of the railway control system, testing this layer is a

key issue. The system is characterized by dedicated hardware with triple redundancy (2-out-of-3) for automatic fault detection. All I/O interfaces offer redundancy with antivalent electrical signals. (Figure 4). Testing in this area has to cope with the following problems:

- Several projects for different railway organizations (specific adaptations, different types of interlocking systems) and different levels of domain knowledge
- Increasing testing costs (often double the planned figure due to an added maintenance effort for a large number of test cases – 3500 system test cases, automated run of acceptance tests for 10 days)
- The requirements specifications are written in a very precise way, but still in human language. As a result, the automated inference of test cases is not straightforward

3.2 The testing approach

For the higher SIL levels 3 and 4, code coverage has to be at least branch coverage, at SIL 4 even MC/DC (Modified Condition Decision Coverage) or LCSJ (Linear Code Sequence Jump). Other recommended testing techniques are equivalence partitioning with boundary value analysis, probabilistic testing, and testing by error guessing. About one iteration per month and one release a year were tested.
Component test cases were specified in the form of word templates and contained a test-case description as well as the file names of the test scripts. The performance as regards response time and memory allocation was validated by load and stress testing in integration tests in a laboratory. The system tests consisted of the validation of all country-specific requirements for the basic system as well as the country-specific components and the logic of the interlocking system. The system-test specifications existed in various forms: (1) in textual form, (2) formalized in the test definition language TDL, (3) a manual test catalog and (4) test cases for automatic execution.
Test cases were traced to requirements using a reference to the system requirements specs by a tool. Extensive reviews and reporting accounted for a major part of the budget. The management of test execution and reporting was the responsibility of the QM. The system tests were the basis for system validation. Only members of QM were authorized to validate the system. The validation report and, implicitly, the results of the system test were part of the Technical Safety Report (Technischer Sicherheitsbericht) and were examined in the course of the approval process by an independent certification authority.
In the case of the EIS, test cases at the system and software level were designed by testers with a thorough knowledge of railway interlocking systems on the basis of specifications written in natural language and of available domain knowledge. Techniques such as equivalence partitioning, boundary value analysis and state transition testing were not systematically applied. Exploratory robustness testing played an important role.
As the following example in Figure 5 shows, some important test cases may be missed, due to an incomplete textual specification. In our case he following has to be clarified: Base state after "current on"? Assumption of the last valid state (H); How is the "suspension defect" state left? Textual requirement specifications – with all references - and test cases are maintained in the test management tool. Each specification has an identifier. To reference the tested requirement (system level or software level), the identifier is inserted in the test and validation documents at system, software and SW-module level. The requirements have two levels of abstraction, a system level and a software level. However, the specifications do not enforce these hierarchies and oscillate from general to very detailed technical descriptions. A lack of formal diagrams also leads to uncertainty about the consistency and completeness of the requirements. So the improvement measure was to use state-transition models or state charts to specify only the risky parts of the dynamic behavior of the interlocking system.



Textual specification:

#REQ-Ka1lsyan_WA_012_BASIS#DEF# for remote-controlled shunts is valid
1. The shunts are set to the right or to the left.
2. If the shunt does not reach its final position within 8 seconds, the current will be automatically turned off (the shunt then has a supervision defect).
3. While a shunt is moving, it must be possible to reverse its movement and reset it to its original position.
4. When a train passes the shunt, the latter has to be locked so that no further movement is possible. The lock has to be opened again after the entire train has passed the shunt.

Figure 5: State chart of an interlocking system

3.3 Results and lessons learned

Systematic test case design methods play an important role when it comes to improving defect detection and reducing quality costs. In the case study, where reliability requirements had high priority, the amount of testing and the resulting costs were major issues. The main cost driver and thus the main disadvantage identified was the number of redundant test cases. The standard IEC 61508 stipulates testing by fault expectation in addition to other testing techniques for software safety validation. Experience-based functional and performance tests were applied. But no tool for probabilistic testing, as recommended in the verification group software safety validation was available at that time. A large number of test suites for automatic module tests were used. However, no tool was available for mutation analysis to evaluate the quality of these test suites. The topics depicted in Figure 6 were identified for improvement and research

ID	Topic	Actual status	Goal
1	Modeling and MBT (model-based testing)	Textual	State-charts, use cases, cause-effect graphing ...
2	Software safety validation	Functional and performance tests	Functional and performance + probabilistic tests
3	Dynamic analysis and test	Structural tests	Structural tests + error seeding
4	Test-case design	Experience-based test-case design	Recommender system for test-case design

Figure 6: Topics of research and improvements

4. Research questions and answers

These topics are currently elaborated in the SoftNet research project under the leadership of the Institute of Software Technology of the Technical University of Graz.

(1) Which techniques for modeling are applicable to complex systems?

As a rule, modeling languages and semantics (meta models) with UML2.0 is one candidate for modeling domain-specific requirements. State charts are used to model embedded systems. We present the IDATG (Integrating Design and Test-Case Generation) tool which already incorporates several powerful testing techniques. [10] The process of modeling and generating test cases is presented in Figure 7. (1) A hierarchical task model is defined in IDATG on the basis of the textual requirements, (2). For each task, the sequence of steps is defined as a task flow. (3) Test data can be imported from a file or be defined manually. However, the most effective way is to use IDATG's data generation algorithms. (4) As soon as a GUI prototype is available, the GUI layout can be recorded with IDATG GUI Spy directly from the screen. The steps of the task flows can be assigned to specific GUI objects. The details of each step are defined. These may include the test instruction, semantic conditions and the test data. (5) IDATG can now generate test cases on the basis of the specification created in the previous steps. All test cases can be converted into various formats. (6) Finally the generated test cases were executed automatically. IDATG is used for testing the Spacecraft Operating System of the European Space Agency ESA, for example. Special attention is given in the IDATG tool to multidimensional equivalence classes in which the partitioning does not depend on only a single input but on a combination of different inputs. The problem here is the possible presence of constraints defining the dependencies of one input on another [12].

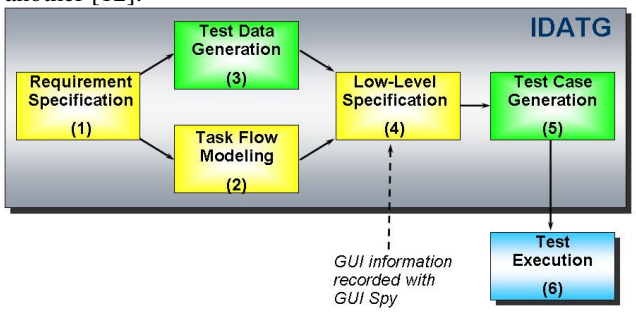


Figure 7: IDATG process

The problem with multidimensional equivalence classes is the inability to visualize them properly. Neither graphs nor tables can offer a comprehensible visualization of higher dimensions. In principle, cause/effect analysis is just another way of visualizing multi-dimensional ECs, although textbooks often treat it as an entirely different method. Causes correspond to input ECs or their sub-conditions while effects correspond to output ECs. This is a promising approach for solving the visualization problem, which is a precondition for test-case generation that is particularly suitable for safety-critical systems. Applying mutation testing shows the effectiveness of these techniques and allows an exact quality statement to be made.

(2) Which probabilistic randomized testing technique for software safety validation is applicable?

To further enhance this approach, randomized testing is to be added. When building blocks are used as a starting point, the new method creates randomly chosen paths through and between these building blocks. These blocks inherently store sequences defined by a test developer [13]. The model of an automated teller machine (ATM) may be

used as an example. The actions are modeled with the IDATG tool as shown in .

In a first step, each of these building blocks is converted to an EFSM. Then, transitions are added between all the EFSMs. Guard conditions are automatically determined from the required state of the GUI elements in the start steps of the corresponding blocks. The EFSM resulting from the above blocks is shown in Figure 8.

For creating randomized test cases (or "random walks"), therefore, these sequences are used rather than just completely random inputs. The conditions for these steps are checked during test-case generation. A problem with pure randomized testing is that certain steps or building blocks may not be used in a random walk because the conditions to get to such a step are too rigid and therefore seldom achieved. To overcome this problem, the generation of randomized test cases is interlaced with a directed search for yet uncovered areas. This combination of randomized testing and a directed search is called a hybrid approach or hybrid algorithm.

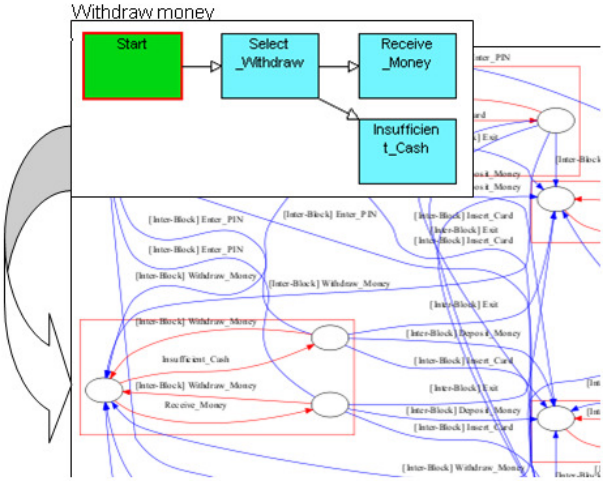


Figure 8: Example ATM – Transformation of IDATG model into an EFSM and an FSM

(3) How could error seeding and mutation analysis be used to check the effectiveness of a module test suite?

The question here is: How well does a test suite perform in detecting defects in the code? Hypothetical assumptions are that a perfect test suite will detect all defects and a perfect implementation passes this test suite. In practice, neither a test suite nor an implementation are perfect. The question is, if a mutant is not detected by a test, then is the test suite ineffective? Program errors are therefore simulated by replacing operators or operands, removing parts of the program etc. to force incorrect behavior. For example, in a program determining the kind of a triangle e.g. equilateral, isosceles etc., the code is mutated in the following way: "if (b == c) return EQUILATERAL" to "if (b <= c) return EQUILATERAL;". A test suite executed against this mutant should detect the error, so the mutant is killed. The mutation score is the ratio between killed mutants and the total number of mutants.

We only consider mutation operators that modify the expression by replacing, deleting, and inserting primitive operators. MuJava provides six kinds of primitive operators; (1) arithmetic operator, (2) relational operator, (3) conditional operator, (4) shift operator, (5) logical operator, and (6) assignment. The MuJava tool was adapted for use to improve component testing in a telecommunications project at Siemens [14].

The mutants are selected by using defect taxonomies, e.g. Beizer [15] or IEEE – categorization using ODC (Orthogonal Defect Classification) [16]. The precondition for mutation analysis is that JUnit-test cases with 80% - 90% branch coverage have to be available (Figure 9).

The benefits are:

- Mutation scores are complementary to coverage-oriented test-completion criteria.
- Development of high-quality software is possible.

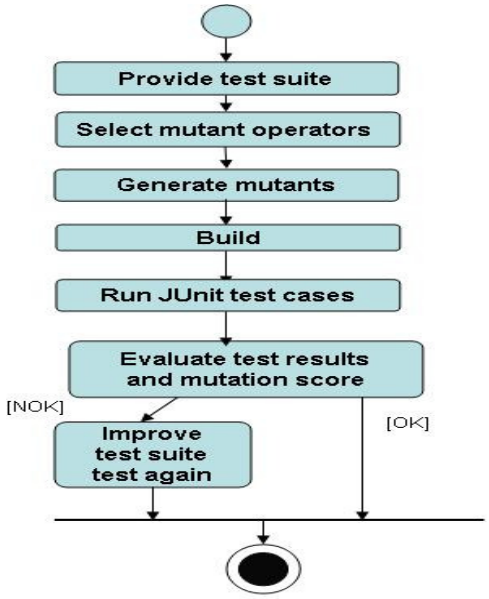


Figure 9: Mutation analysis in component test

(4) What type of recommender system is needed to support the selection of the appropriate testing technique?

Although a wide range of methods and techniques offer guidance for systematic and rigorous testing, the tester's knowledge plays the key role in ultimately and successfully applying these methods and techniques in the context of a project. As the railway-system case study showed, test cases were derived by interpreting the requirements using the knowledge and experience of how railway interlocking systems work in practice. All test cases were developed by testers with years of domain experience and a thorough knowledge of testing railway interlocking systems in order

to overcome any omissions or ambiguities inherent in the natural language specification.
The main cost driver and thus the main disadvantage of this approach identified was the number of redundant test cases. The high total number of test cases resulted in an excessively high testing effort.
Currently available guidelines are of little help to the tester, so the practical use of mind maps, for instance, is very limited. A first step is the characterization schema for testing techniques of Vegas [17]. The requirements for a recommender system are:

- An aid for the interpretation of textual specification and creation of a formal model.
- Context-dependent selection of the appropriate test-case design method.
- Usability aspects with respect to domain experts.

A prototype of a recommender system is currently being developed in SoftNet.

5. Conclusion

Formal modeling, e.g. by using state charts and test-case generation, plays an important role when it comes to improving defect detection and reducing quality costs in verifying safety-critical systems.
The crucial questions that have to be answered are:

- Which modeling and test case design/generation frameworks are suitable for domain experts?
- How efficient is test automation from the view point of reducing testing costs and assuring high product quality?
- How effective are the test cases?

Standards such as IEC61508 permit the use of equivalence partitions/boundary values to reduce the number of test cases. Special attention must be given e.g. to multi-dimensional problems in which the partitioning does not depend on a single input but on a combination of different variables that are interrelated by logical dependencies. The IDATG tool provides a mean for generating more efficient test data by combining the methods of equivalence partitioning/boundary values with cause-effect graphing. Probabilistic or statistical testing, which is prescribed as mandatory in IEC61508 – SIL 4 for software safety validation, was introduced. However, further research is needed to explore the effectiveness of randomized testing with regard to different defect types. Error seeding is recommended as a dynamic analysis and testing technique for SIL 4. A technique for the mutation analysis of Java-Code was presented. Open-source software adapted for the application in a real-world project is currently being evaluated in a large telecommunication project. Further studies are clearly needed to consider the empirical results of various projects in order to select the mutation operators relevant for a specific application area. A recommender system (method guide) for the application of systematic test-case design methods is currently being evaluated to encourage the use of systematic test-case design in testing safety-critical systems. Further pilot projects will be run in future in order to validate the techniques developed here.

6. References

[1] IFIP WG10.4 on Dependable Computing and Fault Tolerance
[2] DO178B: http://en.wikipedia.org/wiki/DO-178B
[3] CENELEC EN 50126 (1999)
[4] CENELEC EN 50128 (2001), EN 50129 (final draft 2002)
[5] B.Peischl: A Survey on Standards and Certification of Safety Critical Software, SNA-TR-2006-01, 2006
[6] R. Black: Advanced Software Testing – Guide to the ISTQB Advanced Certification Vol.1 and 2.; ISBN-13: 978-1-933952-19-2, 2009
[7] S. Ntafos: "On random and partition testing", ACM SIGSOFT symposium on software testing and analysis, 1998
[8] C. Kaner et al.: Testing Computer Software, Int. Thomson Comp. Press, ISBN 1-85032-847-1, 1998
[9] L. Copeland: A Practitioners Guide to Software Test Design Artech House Publ., ISBN 1-58053-991-X, 2004
[10] S. Mohacsi: "Practical Experience with Test Case Generation: Higher Product Quality - Reduced Test Costs ", 1.SoftNet-Workshop „Testen und Verifikation", 7.11.2007 an der TU in Graz, 2007
[11] P. Liggesmeyer: Qualitätssicherung softwareintensiver technischer Systeme, Spektrum Verlag; ISBN 3-8274-1085-1, 2000
[12] A. Beer, S. Mohacsi: "Efficient Test Data Generation for Variables with Complex Dependencies", IEEE ICST in Lillehamer, April 2008
[13] G. Fraser, B.Peischl, F.Wotawa: "A Formal Model for IDATG Task Flows", SNA-TR-2007-P2-03, 2007
[14] Yu-Seung M.J.O.: "Description of Method-level Mutation Operators for Java"; Report George Mason University, 2005
[15] B. Beizer: "Bug Taxonomy and Statistics" Appendix, Software Testing Techniques, second edition, Van Nostrand Reinhold, New York, 1990.
[16] IEEE Computer Society, "IEEE Standard Classification for Software Anomalies", December 2, 1993
[17] S. Vegas: Identifying the Relevant Information for Software Testing Technique Selection; ISESE'04, 2004

Acknowledgement
The research work reported here was partially conducted within the Softnet Austria competence network (www.soft-net.at) and was funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschafts-förderungsgesellschaft mbH. (SFG) and the city of Vienna within the scope of the Centre for Innovation and Technology (ZIT).