A Systematic Approach to Risk-Based Testing Using Risk-annotated Requirements Models

Marc-Florian Wendland, Marco Kranz and Ina Schieferdecker
Fraunhofer Institute FOKUS
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
{marc-florian.wendland, marco.kranz, ina.schieferdecker}@fokus.fraunhofer.de

***Abstract*—Nowadays, software-intensive systems continuously pervade several areas of daily life, even critical ones, and replace established mechanical or manual solutions. Development and quality assurance methods have to ensure that these software-intensive systems are delivered both with adequate quality, optimized resources and within the scheduled time frame. The idea of risk-based testing is to prioritize testing activities to what is deemed critical for the software-intensive system. Although there is a common agreement that risk-based testing techniques ought to be rigorously applied, especially for safety- and security-critical systems, there is actually little knowledge available on how to systematically come to risk-optimized test suites. This paper presents a novel approach to risk-based testing that deals with the transition from risk management and requirements engineering to test design activities and test case generation by using models. The main contribution of the paper is the description of a methodology that allows an easy combination of test generation directives and risk level in order to generate risk-optimized test suites.**

Keywords - risk-based testing; behavior engineering; model-based testing; requirements model; safety-critical systems

I. Introduction

Already in 1999, Amland stated that IT projects are very rarely on time, schedule or budget, so when it comes down to testing, the time to delivery is extremely short and there is no budget left due to the development overrun [5]. This statement holds even for today. This requires test case design techniques to be able to identify the most important test cases to be carried out in view of limited time. Thus, the test cases need to be prioritized to be comparable with each other.

A well-known and highly recommended approach is *risk-based testing* ([24][28][29]). The idea of risk-based testing is as simple as intuitive: Identify prior to test case design and execution those scenarios that trigger the most critical situations for a system in production and ensure that these critical situations are both effectively mitigated and sufficiently tested. Following Bach, risk-based testing aims at testing the right things of a system at the right time [1]. He further states that each test process is actually carried out in a risk-based way due to its sampling characteristics. In most cases, the consideration of risk is rather made implicitly, though.

A *critical situation* is not necessarily dedicated to safety- or security-critical systems (though it is often used in the context of such systems), but applies actually to any kind of system. For example, the most critical situation for a text processor application might be the *save* functionality, since a malfunction may cause the user to switch to the product of a competitor. However, in the area of safety-critical systems, critical situations represent sensitive points in time during the execution of a system, where a malfunction may lead to harm of environment, human life, financial loss etc. No matter what kind of system is tested, the idea of risk-based testing remains the same, whereas the impact of a bad test case selection may differ dramatically, of course.

Even though risk-based testing is deemed helpful to deal with scarce resources, it is a matter of fact that there is only little literature available that provide the tester with a systematic and reproducible approach on how to actually get to a risk-optimized set of test cases. We seek to address the lack of well-founded methodologies for systematic and applicable risk-based testing approaches. Therefore, we are using semi-formal models to describe both the functional-related requirements and a risk-optimized test model. Furthermore, we show how formal test directives are coupled with risk to automate the risk-based test case generation.

The scientific contributions of this paper are:

- Outline of a coherent methodology that combines information from risk analysis and assessment activities with requirements engineering activities
- Use of formal requirements models to incorporate risk-information for further exploitation
- Specify how test case derivation strategies are being coupled to various risk levels in combination with a risk matrix using test directives
- Describe a prototype tooling landscape including complete set of modeling notations for the proposed methodology

However, this paper does not claim to be an industrial evaluation report. The remainder of the work is structured as follows: Section 2 summarizes the state of the art of relevant risk-based-testing approaches to the knowledge of the authors. Section 3 describes the main contribution of this paper, i.e., our methodology for model-based risk-based testing. At first, we give a definition of relevant terms in the realm of risk-based testing. We, then, briefly introduce Behavior Engineering as the basis of our approach. Next, we describe how risk information is incorporated into the resulting requirements models. Afterwards, we show how those risk-annotated requirement models are further exploited for systematic test case generation. Section 4 briefly summarizes a prototype tooling landscape and findings from a first application of the methodology. Section 5 and 6 provide an outlook to future work and conclude eventually.

II. RELATED WORK

The principles of risk-based testing have been addressed by several often quoted publications before (such as [1][2][4]). These articles are mainly dedicated to the clarification of the terms and concepts that belong to risk-based testing. The authors provide justified arguments why risk should always be considered in structuring testing processes. Amland presented a concrete example how risk-based testing had been performed within a project in a financial domain [6]. None of these articles, however, provide precise statements or event suggestions how test design techniques shall be chosen due to an identified and given risk, which is a main contribution of our approach.

Stallbaum and Metzger made a first step towards automated generation of risk-based test suites based on previously calculated requirements metrics (e.g., [8][9]). A prototype research tool called RiteDAP has been presented as being able to generate test cases out of weighted activity diagrams in a two-stage process. At first, paths through the activity are derived in a non-risk based way. Secondly, the paths are ranked due to the risk they include. The traversal algorithm of the test case generator is predefined and not adjustable. The risk-based selection of test cases in that approach is a simple ordering of paths due to their subsumed risk exposures, what might be not sufficient. Our approach, in contrast, envisages that already the traversal algorithm shall be assigned to a certain risk level.

Bauer and Zimmermann have presented a methodology called *sequence-based specification* to express formal requirements models as low-level mealy machines for embedded safety-critical systems (e.g., [10] [11]). By doing so, they build a system model based on the requirements specification. Afterwards, the outcome of a hazard analysis is weaved into the mealy machine. The correctness of the natural language requirements is actually assumed to hold, so that there is no rigorous approach to verify or validate the natural language requirements prior to performing the hazard analysis. Finally, they describe an algorithm that derives test models that include critical transitions out of the system model for each single identified hazard in order to verify the implementation of a corresponding safety function. What they do not present is how to rank the critical transitions in the test models with respect to their risk priority. It is also not clear, whether and how the algorithm they present can be modified in order to vary the test case generation process. Apparently, this approach has ever produced a prototyping tooling beyond research projects.

Kloos has described an approach for transitioning from a fault tree as produced by a fault tree analysis (FTA) [12]. It is used in combination with a system model, expressed as mealy machine, to generate a test model. A test model is in their definition a system model with failure modes and critical transitions leading to the failure modes. As explicitly stated, this approach is dedicated to risk-based testing of safety functions for safety-critical systems. Although the authors claim their methodology to be risk-based, a clear method how the test case generation is actually influenced or guided by the identified risk is not provided.

The most recent approach to risk-based security testing using models is given by Zech for cloud environments [7]. The presented methodology is in a very early state, though. The author claims to fully automate the transition from system models over risk models to misuse cases and eventually to test code. The risk analysis is also planned to be carried out completely automatically by using a vulnerability repository. Neither one of the involved models has been described in greater detail, nor have the involved transformations been specified so far.

Chen discussed an approach for risk-based regression testing optimization [13]. In his approach the author applies a risk value to each test case to prioritize them. Based on these risk values, the test cases are comparable and can be prioritized to either be included in, or excluded from, a re-running regression testing process.

The Behavior Engineering (BE) methodology describes an approach to derive formalized requirements models, so called behavior trees (BT) out of informal, i.e. textual, requirements specifications. It was invented and firstly described in a series of paper (e.g., [17] [18]). Although BE is not related to risk analysis in the first place, we based our approach on it, since we are convinced that the rigorous methodology for requirements formalization is a good starting point to conduct testing in general, and risk-based testing in particular.

Although most of the literature presented above is dedicated to the ideas, principles or theories of risk-based testing, we do see a fundamental lack of concrete methodology on how to integrate the various pieces of information in a systematic way in order to guide the activities of test case derivation. We seek to provide testing experts with comprehensible instructions and a continuous toolset that is based on the principles of model-driven requirements engineering and model-based testing.

III. RISK-BASED TESTING IN A MODEL-BASED WAY

Our approach strives to be generic and applicable for both systems that include functional-related risk mitigation measures (like counter-measures or safety, respectively security functions) and systems that do not include such measures. The latter kind of systems mostly refer to non-safety/security-critical systems, where certain execution paths are deemed critical nevertheless and need intensive testing as well.

In Figure 1, a high-level sketch of our methodology is depicted. In short, the steps are the following:

1. Formalize the requirements specification as integrated behavior tree
2. Augment the integrated behavior tree with risk information
3. Identify for each risk exposure in the integrated behavior tree an appropriate test directive and link them together
4. Pass both the risk-augmented integrated behavior tree and the test directive definition into a test generator

A. *Definitions of risk, risky situation and risk exposure*

Industry-relevant standards (such as [26][28][29]) for systems and software engineering or testing define *risk* (r) as a function of likelihood (l) or probability that a situation occurs during the execution of a system, which is deemed critical, multiplied by the severity (s) of the consequences that may happen if the risky situation is not mitigated, i.e.,

f(R) = l * s.

Beside the term *risk*, there are two other terms denoted in industry standards that apply to the above given definition: *Risk factor* [27] and *risk exposure* [26].

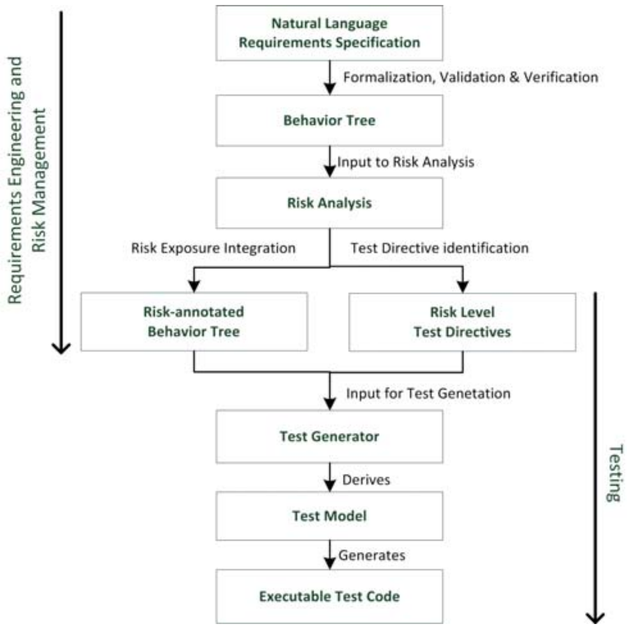
Requirements Engineering and Risk Management
Natural Language Requirements Specification
Formalization, Validation & Verification
Behavior Tree
Input to Risk Analysis
Risk Analysis
Risk Exposure Integration
Test Directive identification
Risk-annotated Behavior Tree
Risk Level Test Directives
Input for Test Genetation
Test Generator
Derives
Test Model
Generates
Executable Test Code
Testing

Figure 1. Overview of the presented methodology

As mentioned before, risk-based testing aims at testing the most critical situations at first and more thoroughly. However, we find that the term critical may cause misunderstandings since people may implicitly think of safety-critical systems. To prevent the reader's confusion, we rather use the term *risky situation* instead. A risky situation may occur in any kind of system (safety-critical or not) and describes a foreseeable sequence of events that leads to a situation where a failure of the systems causes an inacceptable loss (of data, reputation, life, money etc.) [25]. Finally, we use the term *risk* to denote an uncertain event or condition that, if occurs, has a negative effect on the system, and the term *risk exposure* as the comparable value that determines how risky a certain situation really is compared to other situations.

B. *Towards risk-annotated requirements models*

Risk management activities are performed on an information source that provides the risk experts with indications what might go wrong in the system in the field. Each system development project starts usually with a set of requirements that the intended system shall realize. Requirements are normally captured in software/system requirements specifications (SRS) that are structured in a certain way [30]. As already mentioned before, risk analysis activities or simple prioritization considerations are part of almost any development project. At the level of functional-related risk analysis, the SRS deemed to be one of the important information sources, risk experts should take into account. Once the risk analysis has been performed, the risk exposure is usually integrated with the SRS. This leads to a risk-annotated SRS.

Unfortunately, today's requirements specifications are mostly still textual. That entails some inherent problems such as ambiguity due to informal and imprecise textual specifications or the lack of human beings to grasp complex and comprehensive textual specifications. Most recent activities in the realm of requirements engineering strive to employ model-based techniques in order to produce less ambiguous and inconsistent SRSs. As mentioned before, one of these approaches is BE. It is an intuitive yet effective methodology to formalize the functional aspects of natural language requirements for further validation and/or verification activities. BE has been proven extremely beneficial in large-scale industry projects [19]. BE defines two core phases to transform informal requirements into formalized BTs, expressed with the Behavior Modeling Language (BML [36]). The first activity is called *requirements formalization* and provides a well-defined formalization strategy that is, each informal requirement will be translated into a *Requirements Behavior Tree* (RBT). A premise of BE is to stick as close as possible with the vocabulary which was used for expressing the natural language requirement with the advantage of removing any ambiguity being present in the natural language. An RBT comprises the behavioral flow of only one single requirement, namely the requirement it originated from. This keeps the complexity manageable, even of extremely large-scale systems. Hence, the complexity of an entire translation of large-scale requirement specifications is narrowed to the complexity of translating single requirements solely and in a repetitive manner. Myers [37] called this "… an approach with a minimized local problem space that remains constant regardless of the size of the global problem space."

After finishing the formalization of each requirement, the key phase for revealing and detecting flaws in the requirements specification takes place, the *Fitness-for-Purpose* (or simply *integration)* phase. Requirements commonly interact with other requirements, meaning, in a consistent requirements specification without gaps there are intersection points where requirements can be integrated with each other. To identify these points and to continually and rigorously integrate the RBTs with each other is the purpose of the integration phase. Integration is done by seeking parts in the RBTs which are logically identical. In practice, it is often the case that the root node of one RBT occurs elsewhere in another (or multiple) RBTs. Once such potential integration points are identified, the involved RBTs are integrated with each other [17]. The outcome of this activity is the so called *Integrated Behavior Tree* (IBT), expressing a behavioral and compositional overview of the requirements specification of the system. During the integration of RBTs, gaps and ambiguities within the requirements specification can be effectively identified, since a missing or ambiguously stated natural language requirement leads to situation in which an RBT cannot be integrated. If such a situation is detected, an issue has to be recorded and involved stakeholders have to decide how to resolve this situation. Thus, integration aims at improving the quality of the original requirements specification as well as creating an entire overview of the compositional and behavioral intention of the intended system. At this point in time, the requirements specification comprises all information being relevant for the development team to start their activities.

The starting point of our methodology is the outcome of the requirements integration phase with BE, which results in a requirements model expressed as integrated behavior tree (see Figure 1). The requirements model is passed afterwards to the risk experts, which also benefit from the integrated view on the requirements. It allows experts to consider potential failure (no matter if safety/security-critical or not) by traversing or even simulating the behavior of the system captured in a BT. For example, Grunske has presented approaches on how BTs can be leveraged for semi-automated hazard analysis [21]. How the actual risk analysis task is performed is not addressed by our methodology.

Risk analysis copes with risky situations by identifying risk mitigation actions. Risk mitigation may target several aspects of a development project such as organizational, process-related, functional or technical aspects. An organizational mitigation of risk might be to allocate only experienced and certified personnel to stem the project [2]; a process-related one that sufficient testing and manual inspections must be performed, whereas a technical mitigation action might be to rely on well-known and already established software technologies solely. Functional risk mitigation often includes the identification of functional counter-measures that reduce the risk exposure by the system, if correctly implemented. In case there have been functional counter-measures defined that shall mitigate risky situations, they have to be included into the requirements model together with the risk exposure. We end up with a functional-related requirements model that is augmented with risk exposures for risky situations. We call this a *risk-annotated behavior tree (*Figure 2).

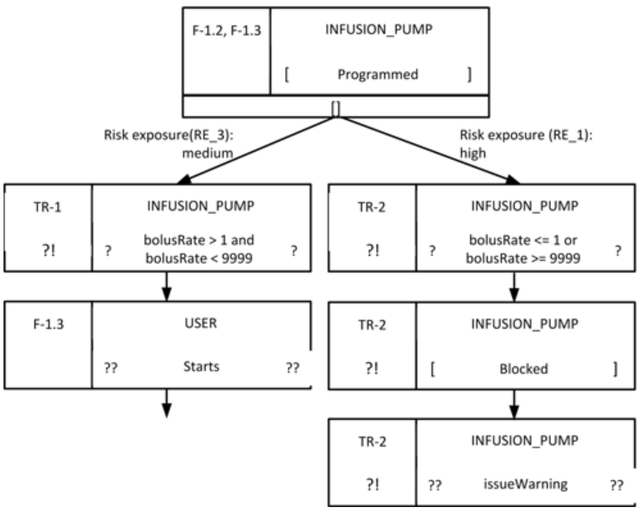

Figure 2. Risk-annotated BT (taken from [31])

C. *The role of risk levels and risk matrices*

An important concept in our methodology of risk-optimized test case generation is *risk level*. Following the ISTQB glossary [28] a risk level indicates the importance of an identified and assessed risk, so it serves the purposes for comparing risks with other risks. Risk levels can be expressed either qualitatively or quantitatively. An often used qualitatively scale for risk levels is *low, medium, high*;however, there is actually no restriction on the number of risk levels being used.

One possible approach, especially in qualitative risk assessment, is to combine risk levels with a risk matrix. A risk matrix is a two-dimensional table for combining likelihood and severity of a risk. An example for risk levels and risk matrixes is shown in Figure 3.

The uppermost table depicts a risk matrix with qualified values for both likelihood and severity (see also [25]) and assigned risk exposures. The middle table assigns cells of the upper risk matrix with risk levels. For the sake of simplicity, we will stick with the aforementioned three risk levels *high*, *medium* and *low*.
Afterwards, the actual risk exposures to risk level assignment can be derived out of the two previous tables.

The result is kind of an instantiation of the risk matrix template plus the assigned risk levels for its cells. Risk exposures that are classified by the same risk level are considered to have an equally negative impact on the system if the corresponding risky situation leads to a system failure. This is shown in the lower table of Figure 3.

		Qualitative severity levels		
		Significant	Moderate	Negligible
Qualtitative probability/ likelihood levels	High	RE_1, RE_4	RE_2	RE_3
	Medium	RE_7	RE_6	RE_5
	Low		RE_8	

		Qualitative severity levels		
		Significant	Moderate	Negligible
Qualtitative probability/ likelihood levels	High	high	high	medium
	Medium	high	medium	low
	Low	medium	low	low

Qualitative risk leves and assignment	
High	RE_1, RE_4, RE_2, RE_7
Medium	RE_3, RE_6
Low	RE_5, RE_8

Figure 3. Risk levels, risk matrix, risk assignment

D. *Systematic test case derivation using test directives*

A test directive is an additional piece of information within the test model that describes precisely what test derivation technique and strategy shall be used by the test generator to generate test cases for a certain risk level. This implies that test directives are bound to risk levels and transitively to the risky situation in the model. Utting [20] provides a good overiew of test derivation strategies for formal, graph-based models such as transition or state coverage.

In risk-based testing, we want to ensure that more risky situations are tested more thoroughly, because they are deemed more critical. It holds true for all risky situations that test cases try to provocate the risky situation to evaluate whether they appear or not. A simple and intuitive interpretation of *more thoroughly tested* for the example depicted in Figure 2 is that we want to ensure that test cases for the risk exposure RE_3 are more elaborated in terms of both structure and data than the ones for RE_3. There are many ways to come to those more elaborated test cases, for example by using different data coverage strategies (e.g., simple equivalence classes for RE_3 in opposite to boundary value analysis for RE_1) or structural coverage criteria (e.g., shortest path into the risky situation RE_3 in opposite to a path that is longer for RE_1). In our methodology, we capture the information on how to derive test cases for a certain risk level (and thus risk exposures) in a test directive. They are the fundamental means in our methodology to enable an automated derivation of risk-optimized sets of test cases as depicted in Figure 1. They make the entire test design activities more systematic, understandable and even more important reproducible. Additionally, the entire test generation process can be easily adjusted to changed needs by just re-defining a test directive's strategy.

The task of defining test directives for certain risk levels is most crucial in our methodology, since it has a crucial impact on the entire risk-based approach. This task requires the intellectual power of experienced personnel in both the current domain and testing. We believe it should not, or even cannot, be performed automatically. However, the actual test case generation approach according to the test directives bears an enormous automation potential. It allows the labor-intensive, error prone and time-consuming manual tasks to be outsourced to an automaton, such as a test case generator.

Eventually, after test case generation has been carried out automatically, a test model is created that contains all the risk-optimized test cases that adhere to the test directive. After execution, the test results analysis takes place. An important outcome of the result analysis is whether the initial assumptions on the risk were properly assessed. In any case, the test results have to be taken into account for a new development cycle, if there is one, in order to adjust the initial assessments with empiric data taken from the test process.

IV. TOOLS AND TECHNOLOGY LANDSCAPE

For the implementation of our methodology, a consequent and integrated tooling landscape and modeling notation are required. In former research projects (e.g., [31]), we identified parts of that tooling landscape. Augmented with the needs for the further elaborated methodology presented in this article, the current tooling landscape and modeling notations are required:

- A language to specify behavior trees based on the Unified Modeling Language (UML) [24] and tooling to perform BE [1]
- Risk extension for behavior trees
- A language and tooling to capture risk matrices, risk levels and testing directives
- A language and tooling to express test models
- A language for executable test scripts

Our own premise for the implementation is to rely on established and well-known technologies and modeling notations instead of reinventing the wheel by using another proprietary solution. We decided to apply UML for all parts of the methodology by using so called *profiles*. A profile is a subset of the UML that adds domain-specific concepts and semantics to UML.

For the BE-related parts of the methodology, we specified a UML profile for the BML (called UBML) that already integrates risk information following the proposed method by Bran Selic [38]. A BT represents the behavioral description of single or integrated requirements. The tree itself is embedded into a surrounding, virtual frame that co-ordinates the process flow. Each node in a BT has a tight interlinking with a component contained in the composition tree, expressing that the behavior exhibited by that node will be executed on the linked component. There is an almost identical diagram type in the UML, namely the *activity diagram*. Activities describe control (and data) flows similar to BTs. Activities are constructed out of actions and edges. An action represents the fundamental and indivisible unit of an executable functionality that may operate on objects. Edges connect actions with each other. Given those ingredients, an activity appears appropriate to be customized for expressing BTs. To keep the analogy with BML, for each behavior node of BML [36] a direct counterpart stereotype has been created in UBML, such as *BehaviorTree, Selection*, *Guard*, *Event.* Structural aspects like components and messages are included, too, partially represented as stereotype (e.g. the stereotype *Message* extends the UML metaclass *Signal*) and partially reusing plain UML (e.g. UML metaclass *Component* and the component diagram are used to model BE components). As example, see the BT expressed as UBML in Figure 4. It is very similar to the original BML notation as depicted in Figure 2.

The test model artifact, as depicted in Figure 1, is expressed with the UML Testing Profile (UTP) [23]. UTP extends UML with test-relevant artifacts, which suits our needs. As test execution language we rely on the Testing and Test Control Notation version 3 (TTCN-3) [35]. All of these technologies are fostered by non-profit organizations (e.g., OMG [32], ETSI [34]), what guarantees vendor and methodological independence as well as continuous maintenance.

We have not yet specified precisely on how to express model risk matrices, risk levels and test directives in UML. In addition, the dependencies among risk exposure (as part of UBML), risk level and test directives have to be established as well. An early implementation of test directive guided test case generation has already been presented [15], and a more elaborated one will be presented in [16]. There is currently an ongoing discussion in the UTP working group [33] whether test directives might be incorporated into the specification. All modeling languages and tooling facilities mentioned above are or will be integrated into our test modeling environment Fokus!MBT, a UTP-based test modeling tool.

V. CONCLUSION AND FURTHER WORK

In that paper, we presented an overview of a noval risk-based testing approach that relies on the principles of model-based testing. Our idea is based on test directives as interpretation of test case derivation techniques that support systematics, transparency and reproducibility of the test derivation task. We do not claim that our methodology is completely automated, because we do believe there is a need for intellectual creativity that can only be carried out manually, even if we rely on the principles of model-based engineering. We doubt the feasibility of just pressing a *magic button* and a risk-optimized set of test cases will be generated automatically. However, there is great potential in expressing suitable key artifacts of a system development process with semi-formal models. It allows capturing the intellectual power of experts in a computer-readable format, so that labor-intensive tasks can be carried out by an automaton.

We are going to apply this approach to more case studies in order to get empirical results for our methodology. What we have done so far was a proof-of-concept, so there have been some lessons learned that have impact on the refinement of the modeling methodology. Another important work to be done is to describe the entire modeling approach on a more technical level in order to explicitly show how things are interconnected with each other semantically and technically.

Further technical work will, in particular, address the target in particular the definition of a precise and stable methodology for doing BE with UBML.

The main focus, however, will be set to the combination and integration of test directives and risk levels, since this is the main contribution of our risk-based methodology and actually the most added value to the current state of the art in the realm of risk-based testing.

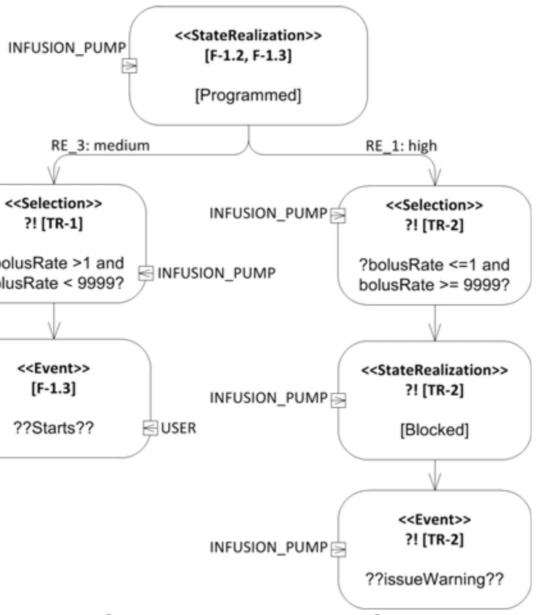

Figure 4. Behavior Tree as stereotyped UML activity diagram

ACKNOWLEDGMENT

This work was partially supported by the projects ROTESS [31] and BTTest (DAAD project number 50449190).

REFERENCES

[1] G. J. Bach, “Heuristic Risk-Based Testing”, Software Testing and Quality Engineering Magazine, November 1999, pp. 96-98.

[2] F. Redmill, “Exploring Risk-based Testing and Its Implications”, 1. Software Testing, Verification & Reliability, 14(1), 2004, pp. 3-15.

[3] F. Redmill, “Theory and practice of risk-based testing”, Software Testing, Verification & Reliability, 15(1), pp. 3-20 2005.

[4] S. Åmland, “Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study”, Journal of Systems and Software 53(3), 2000, pp. 287-295.

[5] S. Åmland, “Risk Based Testing and Metric”, 5th International Conference EuroSTAR 1999, Barcelona, Spain, 1999.

[6] S. L. Pfleegr, “Risky business: what we have yet to learn about risk management”, Journal of Systems and Software 53(3), Elsevier, 2000, pp. 265-273.

[7] P. Zech, “Risk-Based Security Testing in Cloud Computing Environments”, 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST), 2011, pp. 411-414.

[8] H. Stallbaum and A. Metzger, “Employing Requirements Metrics for Automating Early Risk Assessment”, In: Proceedings of the Workshop on Measuring Requirements for Project and Product Success, MeReP07, at Intl. Conference on Software Process and Product Measurement, Spain, 2007, pp. 1-12.

[9] H. Stallbaum, A. Metzger, and K. Pohl, “An Automated Technique for Risk-based Test Case Generation and Prioritization”, In: Proceedings of the 3rd Workshop on Automation of Software Test, AST'08, at 30th Intl. Conference on Software Engineering (ICSE), Germany, 2008, pp. 67-70.

[10] T. Bauer et al., “From Requirements to Statistical Testing of Embedded Systems”, In: Software Engineering for Automotive Systems, (ICSE), 2007, pp. 3-10.

[11] F. Zimmermann, R. Eschbach, J. Kloos, and T. Bauer, “Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems”, In: Proceedings of the 12th European Workshop on Dependable Computing (EWDC), France, 2009.

[12] J. Kloos, T. Hussain, and R. Eschbach, “Risk-Based Testing of Safety-Critical Embedded Systems Driven by Fault Tree Analysis”, In: Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST 2011), IEEE Computer Society, Berlin, 2011, pp. 26-33.

[13] Y , Chen,. R. Probert, and P. Sims, “Specification-based Regression Test Selection with Risk Analysis”, In: Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research (CASCON '02), 2002, pp. 1.

[14] M.-F. Wendland, I. Schieferdecker, and A. Vouffo Feudjio, “Requirements-driven testing with behavior trees”,. In: Proceedings of the Fourth IEEE International Conference on Software Testing, Verification, and Validation Workshops (ICST 2011), IEEE Computer Society, 2011, Germany, 2011, pp. 501-510

[15] M.-F Wendland, J. Großmann, and A. Hoffmann, “Establishing a Service-Oriented Tool Chain for the Development of Domain-Independent MBT Scenarios”, In: Proceedings of 7th Workshop System Testing and Validation (STV’10), IEEE Press, 2010, pp. 329-334.

[16] M.-F. Wendland, M. Kranz, A. Hoffmann, and I. Schieferdecker, “Integration of arbitrary test case generators into UTP-based test models”, 2nd ETSI Model-Based Testing User Conference (MBTUC), Tallinn, Estonia, 2012.

[17] R. G. Dromey, “From Requirements to Design: Formalising the Key Steps” (Invited Keynote Address), In: IEEE International Conference on Software Engineering and Formal Methods (SEFM'03), Brisbane, Australia, 2003, pp. 2-11.

[18] R. G. Dromey, "Genetic Design: Amplifying Our Ability to Deal With Requirements Complexity", in S. Leue, and T.J. Systra, Scenarios, Lecture Notes in Computer Science, LNCS 3466, 2005, pp. 95 - 108.

[19] D. Powell, “Requirements Evaluation Using Behavior Trees: Findings from Industry”. In: Australian Conference on Software Engineering (ASWEC’07), Australia, 2007.

[20] U. Utting and B. Legeard, “PracticalModel-Based Testing – A Tools Approach”. Morgan Kaufmann Publ. (2007)

[21] L. Grunske, “An Automated Failure Mode and Effect Analysis based on High-Level Design Specification with Behavior Trees”, In: Proceedings of International Conference on Integrated Formal Methods (IFM), 2005, pp. 129-149.

[22] K-S. Soon, T. Myers, P. Lindsay, and M.-F. Wendland, “Execution of natural language requirements using State Machines synthesised from Behavior Trees”, The Journal of Systems & Software 85, Elsevier, 2012, pp. 2652-2664.

[23] Object Management Group (OMG): Unified Modeling Language (UML). http://www.omg.org/spec/UML/. Last visit: January 05, 2012

[24] Object Management Group (OMG): UML Testing Profile, Version 1.1 – Beta 1. URL: http://www.omg.org/cgi-bin/doc?ptc/2011-07-19, Last visit:

[25] International Organisation for Standardisation (ISO): ISO:14971 – „Medical devices -- Application of risk management to medical devices“, http://www.iso.org/iso, Last visit: January 05, 2012

[26] International Organisation for Standardisation (ISO): ISO/IEC 16085:2006– Sytems and software engineering, 2006.

[27] International Organisation for Standardisation (ISO): ISO/IEC/IEEE 24765:2010 – Sytems and software engineering, Vocabulary, 2010.

[28] International Software Testing Qualifications Board (ISTQB): ISTQB/GTB standard glossary for testing terms. http://www.software-tester.ch/PDF-Files/CT_Glossar_DE_EN_V21.pdf. Last visit: October 17, 2012.

[29] IEEE Standards Association (IEEE): 829-2008 – IEEE Recommended Practice for Software Requirements Specifications, 2008.

[30] Institute of Electrical and Electronics Engineers (IEEE): 830-1998 – IEEE Recommended Practice for Software Requirements Specifications, 1998.

[31] ROTESS project: Risk-oriented testing of embedded, safety-critical systems. http://www.fokus.fraunhofer.de/de/motion/projekte/laufende_projekte/ROTESS/index.html, last visit: October 17, 2012.

[32] Object Management Group (OMG): http://www.omg.org. Last visit: October 17, 2012.

[33] Object Management Group (OMG) UML Testing Profile Revision Task Force: http://www.omg.org/techprocess/meetings/schedule/UTP.html (access restricted). Last visit: October 17, 2012.

[34] European Telecommunications Standards Institute (ETSI): http://www.etsi.org. Last visit: October 17, 2012.

[35] Testing and Test Control Notation Version 3 (TTCN-3): http://www.ttcn3.org/. Last visit: October 17, 2012.

[36] Behavior Modeling Language (BML): Behavior Tree Notation v1.0, http://www.behaviorengineering.org/docs/Behavior-Tree-Notation-1.0.pdf. Last visit: October 17, 2012.

[37] T. Myers, “The Foundations for a Scaleable Methodology for Systems Design” , PhD Thesis, School of Computer and Information Technology, Griffith University, Australia, 2010.

[38] B. Selic, “A Systematic Approach to Domain-Specific Language Design Using UML”, In: Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), USA, 2007, pp. 2-9.