Automated Test-Design from Requirements - the Specmate Tool

Dietmar Freudenstein, Jeannette Radduenz
Allianz Deutschland AG
dietmar.freudenstein@allianz.de
jeannette.radduenz@allianz.de

Maximilian Junker, Sebastian Eder, Benedikt Hauptmann
Technische Universität München
Qualicen GmbH
{junkerm,eders,hauptmab}@in.tum.de

ABSTRACT

Designing a small set of tests that nonetheless cover the requirements sufficiently is tantamount to keep costs for testing at bay while still maintaining the necessary quality. Engineering an optimal test-suite requires both, insight into the domain and the system under test, but also carefully examining the combinatorics inherent in the requirements. Especially the second part is a cognitive challenge and systematic methods are cumbersome when performed by hand. In this paper, we present Specmate, a tool that supports and partly automates the design of tests from requirements. It provides light-weight modeling techniques to capture requirements, test generation facilities to create test specifications and further supporting functions to derive test procedures or test-scripts from specifications. Specmate has been developed and evaluated in the context of one of the core business systems of Allianz Deutschland, a large insurance company. The source code is freely available at GitHub and an online-demo of Specmate is available at http://specmate.in.tum.de.

1 INTRODUCTION

In this article, we focus on requirements-based testing. Requirements-based testing is a form of black-box testing, where the tests are formulated based solely on the requirements (i.e. without inspecting the software or system structure). The goal of requirements-based testing is to verify that the system does fulfill all its requirements.

The basic process for requirements-based testing consists of three steps. As soon as the requirement is ready for testing, the test-designer analyzes the requirement and starts to develop a test-specification. In this phase he might issue clarification questions to the requirements authors. A test-specification includes details on the logical test-cases that are needed in order to cover the requirements (i.e. to verify that the requirements are fulfilled). The logical test-cases include the input, which should be presented to the system, and the expected outputs. Coming up with a test-specification is potentially difficult as the tester needs to understand the combinatorics behind the requirements. Typically, techniques such as equivalence-class-analysis or cause-effect-analysis are used for creating test-specifications. In the second step the test-designer develops test-procedures from the logical test-cases. Test-procedures provide a step-by-step instruction on how the test-case should be executed. Test-procedures may be formulated as natural-language text or as automatically executable test-scripts. In a final step the tester (or a machine in case of automated testing) executes the test-procedures and identifies any defects.

The test-design (i.e. the creation of test-specifications and test-procedures) influences the quality and the cost of testing. Wrong or incomplete test-cases lead to quality problems in the product. Superfluous tests-cases lead to unnecessarily high costs or missed deadlines in the testing phase. Ideally, a test-specification covers exactly the requirements with as few tests as possible. However, requirements-coverage is only a vague metric if applied to (possibly ambiguous or incomplete) textual requirements.

1.1 Problem

Test-design as described above is a demanding task. Especially for requirements describing complex business rules, engineering a suitable test-specification is not trivial. At Allianz Deutschland, we further identified a gap in the current tool-chain with respect to test-analysis and -design. Hence, test-designers exclusively rely on manual techniques, which come with a high cognitive effort and high error-proneness. Available test-design tools (such as Testona or Conformiq) did not fit our purpose to support test-design for complex requirements. An additional problem is that requirements are often incomplete and ambiguous, lacking the necessary information to derive the right tests from them.

1.2 Contribution

Motivated by the problems described above, we created a tool, Specmate, that supports test-design from textual requirements. In Specmate test-designers can model requirements in a lightweight fashion using Cause-Effect-Graphs (CEG). This modeling step helps to detect ambiguous or incomplete requirements. From the CEG, Specmate can generate test-specifications. From this point the test-designer uses Specmate to derive test-procedures with some further guidance and support. An additional feature of Specmate is the support for deriving end-to-end tests from business process models.

1.3 Related Work

There is a body of work available on test-case generation. Escalona et al. [3] provide an overview over this line of research. One of their conclusions is that while there are many approaches to this topic, there are only few indications if these approaches are useful in an enterprise context, which we target. Regarding test-case generation from cause-effect graphs, several authors discuss different types of algorithms (e.g. [2, 5]), however do not provide a tool. Beer and Mohacsi [1] introduce an method to combine cause-effect analysis with boundary value analysis and integrate the method into a test-design tool. While there are commercial tools in the market, for example Conformiq[1], these tools do either not support creating tests from functional requirements via CEGs or do not support creating end-to-end tests from business processes.

[1]https://www.conformiq.com

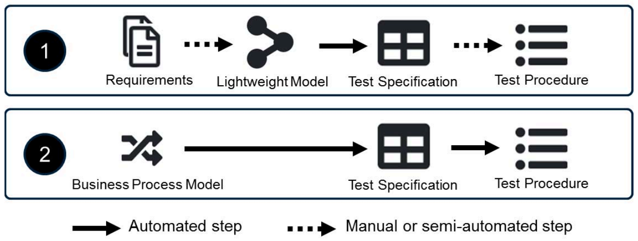


Figure 1: The two workflows currently supported by Specmate.

2 SPECMATE OVERVIEW

Specmate supports creating test-procedures from textual requirements. It integrates into a typical tool landscape with a requirements-management (RM) and test-management tool (TM), importing requirements from RM and exporting test-procedures to TM. Currently, Specmate integrates with HP PPM[2] and HP ALM[3]. Furthermore it can import from a generic text file. Specmate supports two main workflows that relate to two different types of tests.

In the first workflow the input for the test-designer are textual requirements that describe a specific property that the system is required to have. The second workflow relates to end-to-end tests. In this workflow the starting point are models of business processes. From these processes the test-designer then derives end-to-end tests that cover the business process. We will now describe both workflows in more detail in the following sections.

3 REQUIREMENTS-BASED TEST-DESIGN

The first workflow takes textual requirements as an input and produces test-procedures as an output. In Specmate, the test-designer performs three main steps. In the first step, the test-designer models the requirements using cause-effect-graphs (CEG). CEG is a lightweight technique for modeling logical statements and their relationships. In the second step the test-designer generates a test-specification containing logical test-cases from a CEG and adapts the generated test-specification to his need. In the final step, the test-designer then creates test-procedures for the logical test-cases. In the following, we describe the three main steps in more detail.

3.1 Modeling Requirements with Cause-Effect-Graphs

Cause-Effect-Graphs (CEGs) [2] are a graphical model of logical statements and their relationships. A logical statement is represented by a node. Two nodes connected with an arrow mean that the source node is the cause for the for the second node, which is the effect. More than one node can be the causes of an effect node. We differentiate if all causes are necessary to achieve the effect (AND-relationship) or if already one cause achieves the effect (OR-relationship). AND-relationships are graphically represented by a $\wedge$-symbol, OR-relationships are designated by a $\vee$-symbol.

As an example, consider the following (fictitious) requirement of a (fictitious) system for managing car insurances: *A car insurance may be issued if the insurance holder is at least 18 years old and if the registration document for the car is available. If the person is older than 17 years but younger than 18 years, an insurance may be issued if the consent of a parent is available.*

[2]Hewlett Packard Project and Portfolio Management
[3]Hewlett Packard Application Lifecycle Management

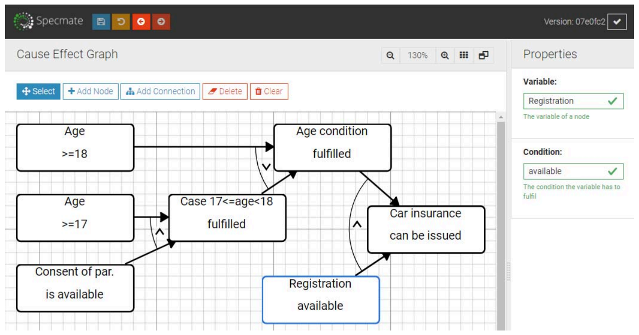


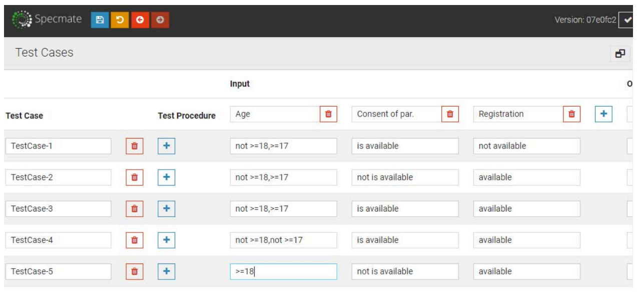
Figure 2: The example requirement modelled as a CEG in Specmate.

Specmate provides a graphical editor to model a CEG for an existing requirement. It supports the basic features of CEGs, such as and-, or- and negation-operations. Figure 2 shows how the example requirement can be represented as a CEG in the Specmate editor.

3.2 Generating Test-Specifications

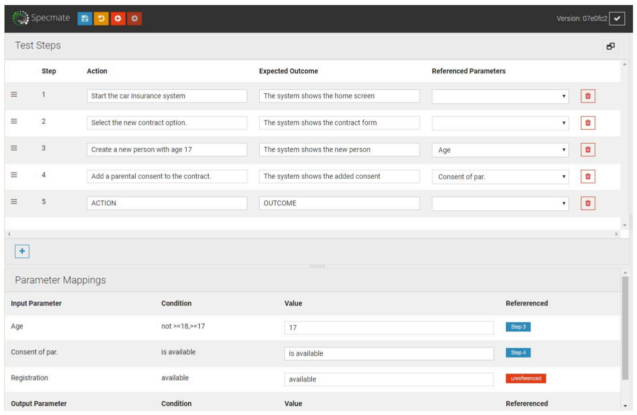
Specmate can automatically generate test-specifications from CEG models. The rules to generate a test-specification from a CEG model are designed to achieve a good probability of finding failures while minimizing the number of test-cases. The rules can be found in [4]. More refined algorithms are proposed, for example, by Paradkar et al. [5]. In short, the CEG graph is traversed from effects back to the causes. For each node with an AND relationship that currently evaluates to *true*, all source nodes are set to *true*. If the node evaluates to *false*, one separate test-case is generated for each source node. In this text-case this source-node is *false*, while all other source nodes are *true*. Hence, for the AND case, this algorithm generates $S + 1$ test-cases, where S is the number of source nodes. The OR case is handled in a similar way and results in the same number of test-cases. In our implementation we use a SAT-solver to find test-cases with the given constraints.

In the example from above, Specmate generates five test-cases: Two for the positive case, testing the situation where the person is above or below the age limit, and three for the negative case. Specmate presents the test-cases in a table. Each row in the table represents a single test-case. Each column in the table is an input or output test-parameter. The test-parameters relate to the nodes in the CEG. Cause-nodes are mapped to input parameters. Effect-nodes are mapped to output parameters. Intermediate nodes are not mapped to columns in the test-case table. Figure 3 shows the test-cases as they are displayed in Specmate. Note that the generated test-specification doesn't need to be the final result. The test-designer has the possibility to edit the specification.



		Input		
Test Case	Test Procedure	Age	Consent of par.	Registration
TestCase-1		not >=18,>=17	is available	not available
TestCase-2		not >=18,>=17	not is available	available
TestCase-3		not >=18,>=17	is available	available
TestCase-4		not >=18,not >=17	is available	available
TestCase-5		>=18	not is available	available

Figure 3: The generated test-cases for the example requirement.



Step	Action	Expected Outcome	Referenced Parameters
1	Start the car insurance system	The system shows the home screen	
2	Select the new contract option.	The system shows the contract form	
3	Create a new person with age 17	The system shows the new person	Age
4	Add a parental consent to the contract.	The system shows the added consent	Consent of par.
5	ACTION	OUTCOME	

Input Parameter	Condition	Value	Referenenced
Age	not >=18,>=17	17	Step 3
Consent of par.	is available	is available	Step 4
Registration	available	available	unreferenced
Output Parameter	**Condition**	**Value**	**Referenenced**

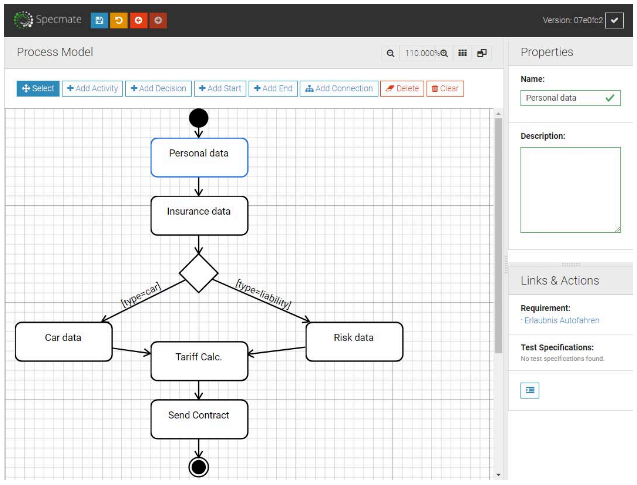
Figure 4: Specmate showing the (partial) test-procedure for a logical test-case. Some of the steps in the upper part reference test-parameters. These parameters are highlighted in blue in the parameter list in the lower part.

3.3 Creating Test-Procedures

The final step for the test-designer is to create test-procedures for the identified logical test-cases. Although Specmate supports the test-designer, this is mainly a manual step. In Specmate, the test-designer models test-procedures as a sequence of test-steps. Each test-step is characterized by a test-action and, optionally, by an expected result of this action. To guide the test-designer, Specmate displays the input and output parameters of the logical test-case for which the test-designer creates the test-procedure. Besides the name of the test-parameter, Specmate displays the values of the parameter in the current test-case.

For each test-step, the test-designer may reference a test-parameter to signal that this parameter has been handled in the test-procedure. Hence, the parameter list works as a checklist, to allow the test-designer to cross-check if the test-procedure is complete.

Figure 4 shows how a test-procedure for the example looks like in Specmate. The displayed test-procedure belongs to the logical test-case *TestCase-3* in Figure 3. It tests the situation where the car insurance can be issued although the person is not yet 18 years old because a parental consent is available. The test-procedure then describes how a tester should proceed, beginning with the start of the system, continuing with the creation of a contract etc. In Step 3 (*Create a new person with age 17*, the test-designer reference the test-parameter *Age*. This reference is also mirrored in the list of parameters (by the blue box beneath the parameter). On the other hand, the parameter *Registration* is not yet referenced (visible by the red box beneath the parameter). This is a hint that the test-procedure is not complete yet.



Name: Personal data

Description:

Requirement: : Erlaubnis Autofahren

Test Specifications: No test specifications found.

Figure 5: Modeling a business process in Specmate

4 END-TO-END TEST-DESIGN

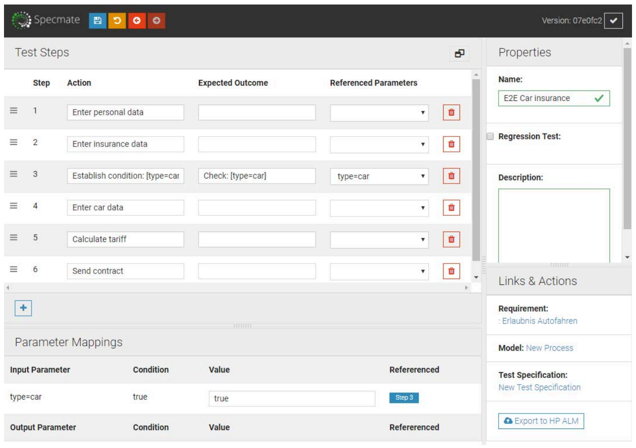
Specmate supports a second main workflow: The design of end-to-end tests based on business process models (BPM). BPMs describe the flow of action needed to reach one or more business goals. For example, a BPM may describe the process to create a new contract, starting from capturing the personal and contract data, continuing with calculating the tariff and finishing with issuing the contract and sending the contract files. Typically a business process involves several actors and systems. An end-to-end test is a test that executes a complete run through a business process. The workflow in Specmate consists of two main steps: First the business analyst models the business process [4]. Afterwards, the test-designer used the modeled business process to generate test-cases from it.

4.1 Modeling Business Processes

Specmate provides a basic graphic modeling language for business processes. It consists of action and decision nodes, as well as connections to model the process flow. Connections flowing out of decision nodes can be annotated with a condition that must be fulfilled in order for the process to continue along this flow. .

Figure 5 shows a simple example BPM as modeled in Specmate. Note that typically business processes are larger and contain several hundred actions. In the example, we model the process of creating a new contract, starting from capturing personal and insurance data. Depending on the insurance type (car insurance or liability insurance) the specific data for this contract type is captured. Afterwards the tariff is calculated and the contract is sent.

[4]Typically, business processes will be already available and only be imported into Specmate


Specmate Version: 07e0fc2

Test Steps

Step	Action	Expected Outcome	Referenced Parameters
1	Enter personal data		
2	Enter insurance data		
3	Establish condition: [type=car	Check: [type=car]	type=car
4	Enter car data		
5	Calculate tariff		
6	Send contract		

Parameter Mappings

Input Parameter	Condition	Value	Referenced
type=car	true	true	Step 3
Output Parameter	**Condition**	**Value**	**Referenced**

Properties

Name: E2E Car insurance

Regression Test:

Description:

Links & Actions

Requirement: : Erlaubnis Autofahren

Model: New Process

Test Specification: New Test Specification

Export to HP ALM

Figure 6: The generated end-to-end test for the case car insurance in the example business process.

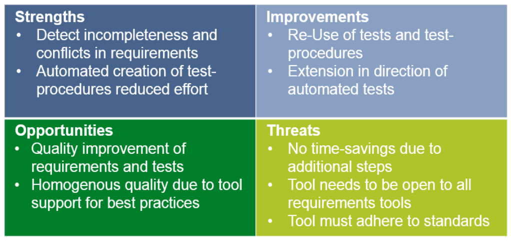
4.2 Generating Tests from Processes

Departing from an BPM, the test-designer can generate end-to-end tests. The goal of end-to-end testing is to cover the business process. The tests generated by Specmate guarantee edge-coverage (or C_1 coverage) with respect to the process graph. That means every connection of the model is part of at least one test-case. As above we distinguish between a test-specification, containing the logical test-cases, and a test-procedure, describing the testing flow step by step. Recall that in the case of CEG models Specmate generates only the test-specification. In the case of BPMs, Specmate generates the test-specification (using the annotated conditions at the edges of the BPM) as well as the test-procedures (using the action nodes).

In the example, Specmate generates two test-cases, one test-case for each of the two insurance types. Figure 6 shows the generated test-procedure for the case car insurance. Each action in the business process is mapped to a step in the test-procedure. For the condition in the BPM, Specmate creates an additional test-step. Note that the test-steps already reference the correct test-parameters.

5 EVALUATION

To evaluate Specmate we conducted a user-study combined with interviews within Allianz Deutschland. In total we included 8 experts and 4 managers from the domains testing and business analysis into our study. We executed the study as follows: With the experts we first conducted a short interview where we asked for challenges and problems in their area. Afterwards we introduced Specmate and presented a couple of tasks that they should solve using Specmate. We asked the participants to think aloud while they were performing the task. During this phase we noted their comments and observed how they interacted with the tool. Afterwards we did another interview where we discussed if and how Specmate addressed the challenges they mentioned earlier, the strengths and weaknesses, chances and risks they see for the introduction of Specmate in the organization. The interviews with the managers were similar, however with a reduced practice part. We summarized the results of the study in form of a SWOT-like analysis.



Strengths • Detect incompleteness and conflicts in requirements • Automated creation of test-procedures reduced effort	**Improvements** • Re-Use of tests and test-procedures • Extension in direction of automated tests
Opportunities • Quality improvement of requirements and tests • Homogenous quality due to tool support for best practices	**Threats** • No time-savings due to additional steps • Tool needs to be open to all requirements tools • Tool must adhere to standards

Figure 7: Main points identified in the user study.

Figure 7 shows the main points of the participants' feedback. They valued the possibility to detect problems in requirements through the modeling and the automated creation of tests. However, they missed functions to re-use parts of tests and test-procedures and a link to test-automation. The participants see opportunities to improve the quality of requirements and tests, especially by applying tool-supported best-practices. The main risks according to the participants are that while quality is improved no time-saving effect occurs, which might hinder adoption. Additionally, as there are several RM tools used in the organization, it is important to interact with those tools.

Apart from the evaluation results, we are aware of further possible limitations that affect Specmate, such as: (1) The requirements in real-live projects are too coarse grained to be used for testing or too complex to be captured with the Specmate models. (2) Real-live business processes use a more powerful syntax.

6 SUMMARY AND OUTLOOK

In this paper we presented Specmate, a tool that supports a structured, semi-automated approach to design tests from requirements. It features two main workflows. One workflow to create tests for single requirements. A second workflow to create end-to-end tests from business processes. We evaluated Specmate in a user-study inside Allianz Deutschland. We found that the participants see Specmate as a tool to improve the overall quality. However, adoption may be at risk if users need to spend excessively much time. Our next steps will be to bring Specmate into production at Allianz Deutschland within three pilot teams to gain insight into the performance of the tool in practical use. We will investigate specifically which limitations Specmate has when employed in real-live projects We value feedback of the scientific community: Therefore the source code of Specmate can be accessed at http://www.github.com/junkerm/specmate. An online demo is available at http://specmate.in.tum.de.

REFERENCES

[1] A. Beer and S. Mohacsi. 2008. Efficient Test Data Generation for Variables with Complex Dependencies. In *2008 International Conference on Software Testing, Verification, and Validation.*

[2] William R Elmendorf. 1973. *Cause-effect graphs in functional testing.* IBM Poughkeepsie Laboratory.

[3] M.J. Escalona, J.J. Gutierrez, M. Mejías, G. Aragón, I. Ramos, J. Torres, and F.J. Domínguez. 2011. An overview on test generation from functional requirements. *Journal of Systems and Software* 84, 8 (2011), 1379 – 1393.

[4] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing.* John Wiley & Sons.

[5] Amit Paradkar, K.C. Tai, and M.A. Vouk. 1997. Specification-based testing using cause-effect graphs. *Annals of Software Engineering* 4, 1 (Jan 1997).