Software Qual J
https://doi.org/10.1007/s11219-018-9407-9

Comprehensibility of system models during test design: a controlled experiment comparing UML activity diagrams and state machines

Michael Felderer[1,2] · **Andrea Herrmann**[3]

Abstract UML activity diagrams and state machines are both used for modeling system behavior from the user perspective and are frequently the basis for deriving system test cases. In practice, system test cases are often derived manually from UML activity diagrams or state machines. For this task, comprehensibility of respective models is essential and a relevant question for practice to support model selection and design, as well as subsequent test derivation. Therefore, the objective of this paper is to compare the comprehensibility of UML activity diagrams and state machines during manual test case derivation. We investigate the comprehensibility of UML activity diagrams and state machines in a controlled student experiment. Three measures for comprehensibility have been investigated: (1) the self-assessed comprehensibility, (2) the actual comprehensibility measured by the correctness of answers to comprehensibility questions, and (3) the number of errors made during test case derivation. The experiment was performed and internally replicated with overall 84 participants divided into three groups at two institutions. Our experiment indicates that activity diagrams are more comprehensible but also more error-prone with regard to manual test case derivation and discusses how these results can improve system modeling and test case design.

Keywords UML models · System testing · System models · Test design · Model comprehensibility · Controlled experiment

✉ Michael Felderer
michael.felderer@uibk.ac.at; michael.felderer@bth.se

Andrea Herrmann
herrmann@herrmann-ehrlich.de

[1] University of Innsbruck, Innsbruck, Austria

[2] Blekinge Institute of Technology, Karlskrona, Sweden

[3] Herrmann & Ehrlich, Stuttgart, Germany

Published online: 23 April 2018

1 Introduction

Behavior models like UML activity diagrams or state machines are not only used for system modeling, but are also a valuable basis for deriving test cases in model-based testing (Utting et al. 2012). Model-based testing is a variant of system testing that relies on explicit models that encode the intended behavior of a system under test (SUT) (Utting et al. 2012). As indicated in an experiment by Pretschner et al. (Pretschner et al. 2005), both automatically and manually derived model-based test suites can detect significantly more requirements defects than handcrafted test suites that were directly derived from the textual requirements. The fully automated derivation of test cases from UML diagrams is still difficult and challenging. This is due to the fact that high-quality UML models, which contain all information needed for automatically deriving test cases, are required for this purpose. However, such models are rarely available in practice. In addition, there is empirical evidence (Pretschner et al. 2005) that handcrafted model-based suites detect as many errors as automatically generated model-based test suites with the same number of tests. Therefore, we investigate the case which is still practically more relevant than automation: A test designer analyzes a UML activity diagram or state machine and derives several test cases manually in order to achieve test coverage. Like all complex manual activities, this kind of test case derivation is error-prone and requires comprehensible test models that require a significant amount of time to be created (Mohacsi et al. 2015) and impact the quality of the derived test cases.

The objective of the study presented in this paper therefore is to examine model comprehensibility when manually deriving test cases from UML activity diagrams and state machines, and whether there are differences between these diagram types. For this purpose, three measures for comprehensibility are collected from each diagram type in a controlled experiment: (1) the self-assessed comprehensibility, (2) the actual comprehensibility measured by the correctness of answers to comprehensibility questions and (3) the number of errors made during test case derivation. Both serve as system models from behavioral perspective and are in practice often used as alternatives (Pohl and Rupp 2011). In a previous publication (Felderer and Herrmann 2014), we have analyzed the types of errors made when manually deriving test cases from UML models. In this publication, the comprehensibility aspects of manually deriving test cases from activity diagrams and state machines are investigated.

Knowing which of both diagram types better serves the purpose of test case derivation can thus be useful for practice, when one has to select the right UML model type for system modeling taking testing aspects into account. As both types of models, i.e., UML activity diagrams and state machines, are often used interchangeably in practice to represent the system’s behavior as well as use cases of a system (Pohl and Rupp 2011) and to further derive test cases, it is relevant to investigate which model type is more comprehensible during test design. In industry, it is common to have system tests executed by test personnel with some domain knowledge, but only little experience in systematic testing (Felderer et al. 2014) (Felderer and Herrmann 2014), for instance by key users. The required skills in test design are then often provided in short trainings (Felderer et al. 2014) (Felderer and Herrmann 2014). This situation is similar to courses at university if domains familiar to students and suitable trainings are provided. We therefore investigate the difference between the two diagram types, i.e., UML activity diagrams and state machines, in a controlled experiment with overall 84 students divided into three groups at two institutions, i.e., the experiment was performed with two groups at Duale Hochschule Baden-Württemberg in Karlsruhe (Germany) and its replication (Mendonça et al. 2008) by the same researchers was performed at the University of Innsbruck (Austria).

This paper is structured as follows. Section 2 gives an overview of related work. Section 3 covers the experiment description and Section 4 presents the experiment results and their analysis. Section 5 discusses the results and Section 6 threats to validity. Finally, Section 7 concludes this paper and presents future work.

2 Background and related work

Comprehensibility of UML models in the context of manual derivation of test cases have not been investigated empirically before. However, there are two types of related work: (1) empirical studies on comprehensibility of UML models discussed in Section 2.1 as well as (2) on the manual derivation of test cases discussed in Section 2.2. In addition, in Section 2.3 we sketch the background on system test cases, as in our study we derive system test cases from system models. In the remainder of this paper, we use the terms “UML activity diagrams” and “activity diagrams” as well as “UML state machines” and “state machines” interchangeably.

2.1 Empirical studies on the comprehensibility of UML models

Quality of UML models has been investigated intensively in recent years and several empirical studies are available (Budgen et al. 2011; Fernández-Sáez et al. 2013). The most empirically studied quality attribute of UML models is comprehensibility (Cruz-Lemus et al. 2011a). It has been investigated from different perspectives. Some approaches focus on layout and visualization aspects of UML models (e.g., (Purchase et al. 2001; Wong and Sun 2006; Eichelberger and Schmid 2009; Sharif and Maletic 2010; Störrle 2012; Störrle 2014)). Others study styles and rigor in UML models (e.g., (Briand et al. 2005; Nugroho 2009)) or look into the effect of using stereotypes on model comprehension (e.g., (Staron et al. 2006; Sharif and Maletic 2009; Ricca et al. 2010; Cruz-Lemus et al. 2011b)). Finally, closest to our work, different UML diagram types are compared (Otero and Dolado 2004; Glezer et al. 2005). In two controlled student experiments, Otero and Dolado (Otero and Dolado 2004) compare sequence diagrams and state machines. They observe that (1) state diagrams provide higher semantic comprehension of dynamic modeling in UML when the domain is real-time and sequence diagrams are better in the case of a management information system, and (2) regardless of the domain, a higher semantic comprehension of the UML designs is achieved when the dynamic behavior is modeled by using both sequence diagram and state machine. Results of a controlled student experiment by Glezer et al. (Glezer et al. 2005) who compare collaboration and sequence diagrams indicate that collaboration diagrams are easier to comprehend than sequence diagrams in real-time systems, but there is no difference in comprehension of the two diagram types in management information systems. The difference between UML activity diagrams and state machines has not been compared in controlled experiments, neither with respect to comprehensibility nor with respect to manual test case derivation. Comprehensibility (also called understandability) in different studies is measured by

- time need to read the diagram and to answer comprehensibility questions (Otero and Dolado 2004; Glezer et al. 2005; Aranda et al. 2007),

- self-assessed perceived comprehensibility or confidence (Glezer et al. 2005; Aranda et al. 2007; Cioch 1991),
- the number or proportion of correct answers to comprehensibility questions (Otero and Dolado 2004; Aranda et al. 2007; Cioch 1991; Agarwal et al. 1999; Genero et al. 2008; De Lucia et al. 2010),
- understandability efficiency, i.e., number of correct answers to comprehensibility questions divided by time need (Genero et al. 2008; Cruz-Lemus et al. 2005),
- recall, i.e., proportion of the correct answers given by the participant out of all correct answers, precision, i.e., proportion of the correct answers given by the participant out of all answers given, and the F-measure (= 2 (precision recall)/(precision + recall)) for multiple choice questions asked (De Lucia et al. 2010; Gravino et al. 2008).

Aranda et al. state that there are many variables to consider, and it may not be feasible to evaluate them all in a single empirical study (Aranda et al. 2007). The most frequently used comprehensibility metric is the number of errors made when answering comprehensibility questions. So, we will use it in our experiment too.

2.2 Empirical studies on manually deriving test cases from UML models

This subsection provides a short overview on the state of the art of manually deriving test cases from UML models. First, automated model-based testing is an approach claimed to improve testing effectiveness although there is not much evidence for this. Pretschner et al. (Pretschner et al. 2005) evaluate model-based testing based on a case study of an automotive network controller. It motivates the usage of models on the system level for deriving tests. Automatized test case derivation and manual test case derivation seem to be equally efficient in terms of error detection. Many researchers have shown that it is possible to automatically derive test cases from an activity diagram or state machine, and to obtain as many test cases as are needed for test coverage (see (Briand and Labiche 2002a; Kundu and Samanta 2009; Linzhang et al. 2004; Mingsong et al. 2006) for activity diagrams as well as (Offutt and Abdurazik 1999; Kim et al. 1999; Riebisch et al. 2003; Samuel et al. 2008) for state machines). However, for automatic test generation, complete system models are needed, while during requirements engineering and system design, the system models are often modeled just good enough to agree about the functionality with the stakeholders. Therefore, information which is intuitively clear to the project team might be not modeled, several steps might be summarized for more clarity and not all special and error cases might be drawn. Some information like preconditions and other conditions must be specified explicitly by enhancing the models by test-specific stereotypes or OCL expressions (Object Constraint Language). Both variants may be complex and hard to perform for a practitioner. A human test designer, however, when deriving test cases, might be able to add information which is not explicitly modeled. For sure, automatic generation may to some extent still be doable with incomplete system models that may even be refined later in the development process. However, this also requires additional effort and expertise for implementing the test automation which is often not available.

Many empirical studies have investigated testing techniques (Juristo et al. 2004) or compare different defect detection methods like testing and inspection (Runeson et al. 2006). Juristo et al. (Juristo et al. 2004) summarize 25 years of empirical research on software testing techniques based on more than 20 studies and conclude that collective knowledge on testing techniques is limited. Runeson et al. (Runeson et al. 2006) provide a survey on empirical defect detection studies comparing inspection and testing techniques based on evidence reported in 10 experiments and 2 case studies. Both techniques have low effectiveness; reviewers find only 25 to 50% of an artifact's defects using inspection, and testers find 30 to 60% using testing. Using test models is one approach claimed to improve effectiveness, although there is hardly any evidence for this. Pretschner et al. (Pretschner et al. 2005) present an evaluation of model-based testing and its evaluation based on a case study of an automotive network controller. Both automatically and manually derived model-based test suites detected significantly more requirements errors than handcrafted test suites that were directly derived from the requirements. The number of detected programming errors did not depend on the use of models. Automatically generated model-based test suites detected as many errors as handcrafted model-based suites with the same number of tests. Finally, Nugroho and Chaudron (Nugroho and Chaudron 2014; Nugroho and Chaudron 2009) investigate the effect of UML modeling, specifically the production of class and sequence diagrams, on the quality of the code, as measured by defect density, and on defect resolution time in the context of an industrial Java system. The authors found that the production of UML class diagrams and sequence diagrams reduces defect density in the code and the time required to fix defects.

Granda et al. (Granda et al. 2014) measure a test case's quality by semantic completeness: "Semantic Completeness means that it contains all the statements about the domain that are correct and relevant" (see also (Lindland et al. 1994)) and "We measure this variable by counting how many functions expressed in the input specification are also treated in the abstract test cases" (Granda et al. 2014).

In our previous study (Felderer and Herrmann 2014), we analyzed the quality of test cases manually derived from UML activity diagrams and state machines, which resulted in a taxonomy of error types in test cases and their frequencies. In this study, the comprehensibility of UML activity diagrams and state machines themselves during manual test case derivation is investigated.

2.3 System test cases

In our study, we derive system test cases for requirements from requirements models. These are black box tests in the sense of the following definition: "Black-box testing describes testing based on external specifications. It observes the operations to be executed. Therefore, input data determine the appropriate action status of the program, and an output data sequentially executed after invoking an input data." (Pretschner et al. 2005). We have slightly adapted the test case template of the International Software Testing Qualifications Board (ISTQB) (ISTQB 2012). Our tabular template for test cases, which is also used in industrial projects (Felderer and Beer 2013), is shown in Table 1. An identifier is inserted in the upper left corner next to the precondition. A test case consists of a precondition as well as a sequence of test steps. A test case contains a sequence of test steps, each with operation call, input data, and expected results developed for a particular objective or test condition, such as to exercise a particular path or to verify compliance with a specific requirement. A test step is mainly determined by its operation call which further defines the input data and expected result. Each test step is documented as one numbered row including an operation call, input data, and expected result(s). We skip an explicit postcondition which in our template is included in the expected result of the last test step.

Table 1 Test case 1 for a gumball machine (which we used for training the participants)

TC_1	Precondition: matching coin inserted		
No.	Operation call	Input data	Expected result
1	Turn lever		Coin thrown in
2	Check coin Number		Not enough coins inserted
3	Insert coin	Matching coin	
4	Turn lever		Coin thrown in
5	Check coin Number		Enough coins thrown in
6	Eject gumball		Gumball ejected

3 Experiment description

This section is structured as follows. Section 3.1 presents the research questions and analysis procedure, Section 3.2 the experiment design, and Section 3.3 the execution of the experiment.

3.1 Goal, research questions, and variables

The goal of this experiment was to understand the model comprehensibility during the process of manual test case derivation from UML models. The two diagram types (DT) activity diagrams (AD) and state machines (SM) were compared to each other. We used models of two different systems: an automated teller machine (ATM) and a drink vending machine (DVM). For this purpose, the following research questions are investigated:

> *(RQ1) Are there differences between models described as UML activity diagrams and state machines with regard to errors made when manually deriving test cases?*
> *(RQ2) Are there differences between models described as UML activity diagrams and state machines with regard to self-assessed comprehensibility when manually deriving test cases?*
> *(RQ3) Are there differences between models described as UML activity diagrams and state machines with regard to actual comprehensibility when manually deriving test cases?*
> *(RQ4) Does the self-assessed comprehensibility of the models correlate with the number of errors made when manually deriving test cases?*
> *(RQ5) Does the actual comprehensibility of the models correlate with the number of errors made when manually deriving test cases?*
> *(RQ6) Does the self-assessed comprehensibility of the models correlate with the actual comprehensibility?*

The independent variable for all research questions is the diagram type (DT), i.e., UML activity diagram (AD) or UML state machine (SM). Figure 1 shows the dependent variables which we consider in the research questions and their relationships to these: the number of errors (NOE) made in the derived test cases, the self-assessed comprehensibility (SCOMP), and the actual comprehensibility (ACOMP). The research questions assigned to specific variables in Fig. 1 (i.e., RQ1, RQ2, and RQ3) are answered by hypothesis testing, and the research questions assigned to relationships in Fig. 1 (i.e., RQ4, RQ5, and RQ6) are answered by correlation analysis between the two related variables (see Section 3.4 for further details regarding the analysis procedure).

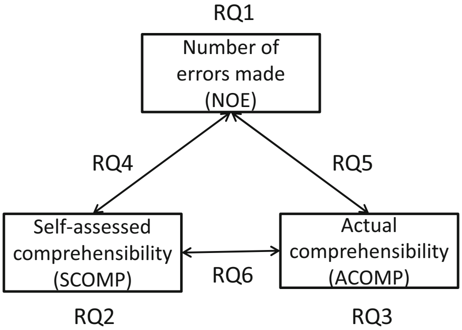
RQ1
Number of errors made (NOE)
RQ4
RQ5
Self-assessed comprehensibility (SCOMP)
RQ6
Actual comprehensibility (ACOMP)
RQ2
RQ3

Fig. 1 The relationship between research questions RQ1 to RQ6 and used dependent variables to answer them

In the context of this study, we define an error in a test case as follows: A test case contains an error when it deviates from the sample solution in a way that information is missing, too much, too detailed or not detailed enough, or when it is written to the wrong template field. For instance, for the test case shown in Table 1, deviations would be that one of the six test steps is missing, or that the input data “matching coin” is cited as expected result. Semantic completeness is most important here, as a test case differing from the sample solution can be semantically correct, too. In order to count errors, we compared the test cases created by the experiment participants to our sample solutions. These were the four sets of test cases needed for branch coverage for each of the four diagrams. The sample solution contained three test cases for the ATM’s activity diagram and four for the DVM’s activity diagram. Both state machines demanded only one test case for branch coverage by traversing loops several times. We distinguish between three levels of errors, i.e., test step, case or suite:

- Test step errors (NOE_STEP): errors with regard to input data, expected result or a complete test step. Their number is counted individually, so each test case can contain several step errors.
- Test case errors (NOE_CASE): errors made systematically in a whole test case for all steps or errors with regard to the precondition
- Test suite errors (NOE_SUITE): errors made in all test cases of the test suite

Errors with regard to input data, expected result or the overall test step are counted on the level of test steps, i.e., for each row in a test case description as shown in Table 1. Errors with regard to precondition or the overall test case are counted on the level of test cases, i.e., zero or one time for each test case as shown in Table 1. Finally, errors with regard to the test suite (i.e., errors which appear in all test cases of one test suite) are counted on the level of test suites. If a specific error affects all test steps, then it counts on the level of test cases, and if an error affects all test cases (including their precondition), then it counts on the level of the test suite. For instance, if error “precondition missing” holds for all test cases of a test suite (with more than one test case), then the error “precondition missing” is counted for the overall test suite. If input is missing for all test steps of a test case (with more than one test step), then the error “no input” is counted for the overall test case. All analyses of errors to answer RQ1, RQ4, and RQ5 are performed for NOE_STEP, NOE_CASE as well as for NOE_SUITE.

The self-assessed comprehensibility (SCOMP) of a diagram is the participations’ subjective rating of comprehensibility measured on a five-point Likert-scale with values “very good”, “good”, “average”, “bad”, and “very bad.”

The actual comprehensibility (ACOMP) of a diagram refers to the answers to four comprehensibility questions about the diagram. These questions were open free text questions (no multiple choice) and had only one correct answer. An answer could be wrong (0 points) or correct (1 point), but also partly correct answers were possible and counted half a point. We assume: If the participant gives a wrong answer to these questions, this shows that he has not well understood this aspect of the diagram. ACOMP is measured by the sum of points achieved when answering the four comprehensibility questions.

3.2 Experiment design

The experiment and its replication was performed with students as experimental subjects. Three groups of students participated at two institutions, i.e., two groups of Business Informatics students at the Duale Hochschule Baden-Württemberg in Karlsruhe (Germany) as well as a group of Computer Science students at the University of Innsbruck (Austria). The participants were additionally motivated to perform well by the fact that the manual test case derivation was also part of the final written exam and the experiment was therefore a good training for them. The groups were named after their institution, i.e., Karlsruhe group 1, Karlsruhe group 2, and Innsbruck. All students in each of the groups knew UML models from previous courses. At the Duale Hochschule Baden-Württemberg, the students attended a one-semester course on software analysis where they learned about the different UML diagram types including activity diagrams and state machines before the experiment actually took place. Additionally, they applied UML models in an extensive one-semester case study project. At the University of Innsbruck the situation was similar: the students attended a one-semester course on software development and project management where they also learned about the different UML diagram types including activity diagrams and state machines. In parallel, the students applied UML in an accompanying practical one-semester software project. All students received the same introductory training and domains familiar to them were chosen, i.e., a gumball machine in the training phase as well as a drink vending and an automated teller machine in the actual experiment. Besides basic knowledge about the applied UML models, neither specific business nor technical knowledge was required with respect to the main task to execute during the experiment, i.e., the manual derivation of test cases from activity diagrams and state machines. As it was therefore unlikely that the Business Informatics and Computer Science students have different prerequisites with respect to executing the experiment task, each group performed the same one. We tested our assumption that there are no differences between the three groups by a Kruskal-Wallis one-way analysis of variance (Crawley 2012) which confirmed that the perceived comprehensibility and number of errors made for activity diagrams and state machines do not differ significantly between the three groups. This context is similar to how system testing is often performed in industry, where it is not uncommon to have system tests executed by test personnel with some domain knowledge, but only little experience in systematic testing (Felderer et al. 2014). For instance, when implementing enterprise systems, testing is often performed by key users who are no software engineers, but future users of the system. The required skills in test design are then often provided in short trainings, i.e., similar to the training phase in our experiment. Such an experiment with unexperienced participants can show what is difficult about a task, i.e., in our case, the manual derivation of test cases from UML activity diagrams and state machines, and where a procedure is not intuitive. We did not prescribe any procedure for the concrete test case derivation, but only investigated the derived test cases themselves.

The main task for each participant was to *derive test cases from activity diagrams and state machines in order to achieve branch coverage*. This means that all transitions (between activities or states respectively) or edges of the diagram must be called by at least one test case. All students had to perform this task twice with varying domains, i.e., drink vending machine (DVM) and automated teller machine (ATM), and diagram type, i.e., UML activity diagram (AD), and state machine (SM). The activity diagrams used in the experiment contained 12 activities (each, for DVM and ATM). The state machines had three states (the DVM) respectively five states (the ATM). These diagrams were developed for this experiment and validated by persons with industrial experience. Their complexity was limited due to the natural time constraints of the experiment. In order to avoid learning effects which would result in better test cases for the model used second, each group used each system only once. This resulted in two treatments: In treatment A, first test cases where derived from the AD of the DVM, and as second task from the SM of the ATM, and in treatment B first from the SM of the DVM and second from the AD of the ATM (see Table 2). In both treatments, the DVM model was used first. Table 2 provides an overview of the experimental treatments per group.

For each group, we applied a *between-subjects balanced design* (Kirk 1995) in which each treatment (A and B) has an equal number of participants as experimental subjects. The assignment of the students to one of the two treatments was performed randomly. The design of the experiments is robust against carryover effects as both the domain and the diagram type change between the tasks of each treatment and avoid carryover in both respects when deriving test cases.

For each task, the students received two sheets. The first sheet included the instructions, the diagrams, and comprehensibility questions. The instruction for test case derivation read as follows: “Please write down all test cases, which you need for achieving branch coverage. For each test case, define the preconditions, individual test steps, input data and expected results. Enter the test cases in the template. Try to define test cases with the possible minimum of test steps.” We did not prescribe any procedure for the test case derivation. The second sheet contained the empty tabular test case templates (like in Table 1) to be filled by the students.

The activity diagrams contain lifelines, activities and decision nodes, one initial node and final nodes. Each decision node has several outgoing edges with disjunctive conditions. Figure 2 shows an example for such an activity diagram modeling the flow of the gumball machine used in the training phase. This activity diagram comprises five activities and has a cyclomatic complexity (McCabe 1976) of 3 (with 11 nodes and 12 edges). The activity diagrams used in the experiment were slightly larger and comprised 12 activities, both for DVM and ATM. Their cyclomatic complexities were 2 for the DVM (with 19 nodes and 19 edges) and 3 for the ATM (with 19 nodes and 20 edges).

Table 2 Overview of treatments per group: for each group, treatment A means that the participants first derived test cases from the DVM’s AD and then from the ATM’s SM, and treatment B vice versa first from DVM’s SM and then from ATM’s AD

	Karlsruhe group 1	Karlsruhe group 2	Innsbruck
Treatment A	DVM-AD ATM-SM	DVM-AD ATM-SM	DVM-AD ATM-SM
Treatment B	DVM-SM ATM-AD	DVM-SM ATM-AD	DVM-SM ATM-AD

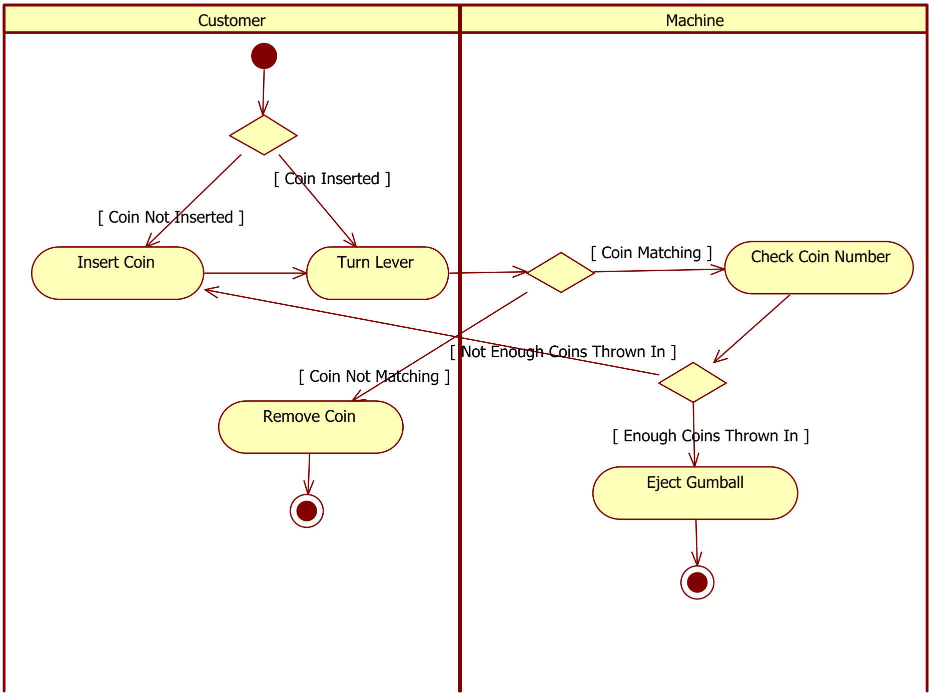

Fig. 2 Test model of gumball machine described as UML activity diagram used to prepare the participants in the training phase

The state machines contain states and transitions with triggers, guards, and events. Figure 3 shows such a state machine modeling the flow of the gumball machine used in the training phase. This state machine comprises four states and has a cyclomatic complexity (McCabe 1976) of 5 (with 6 nodes and 9 edges). The state machines used in the experiment had 3 states for the DVM and 5 states for the ATM. Their cyclomatic complexities were 5 for the DVM (with 5 nodes and 8 edges) and 6 for the ATM (with 7 nodes and 11 edges).

The input and output of an activity usually was obvious based on the model (in the activity diagrams) or modeled explicitly (in the state machines). In the gumball example, we have the activity “insert coin” and later-on the condition “coin matching”, from which the tester can conclude that the input to step “insert coin” is a matching coin and the expected result is “coin inserted.” We took care that both models of the same machine were equivalent in terms of level of abstraction as well as content, although the activity diagram notation originally is meant as a system’s black box model and the state machine a white box model.

In our experiment, test cases are supposed to be executed manually and on the system level. According to our practical experience of the structure of test cases, we adapted the ISTQB definition (ISTQB 2012), as described in Section 2.3. A test suite is a set of several test cases for the same system under test.

We specify each test case in a tabular template. The template for test cases, which is also commonly used in industrial projects (Felderer and Beer 2013), is shown in Table 1, Table 3, and Table 4. An identifier is inserted in the upper left corner next to the precondition. Each test step corresponds to one numbered row including an operation call, input data and expected result(s). All activities of an activity diagram as well as all triggers and events of a state

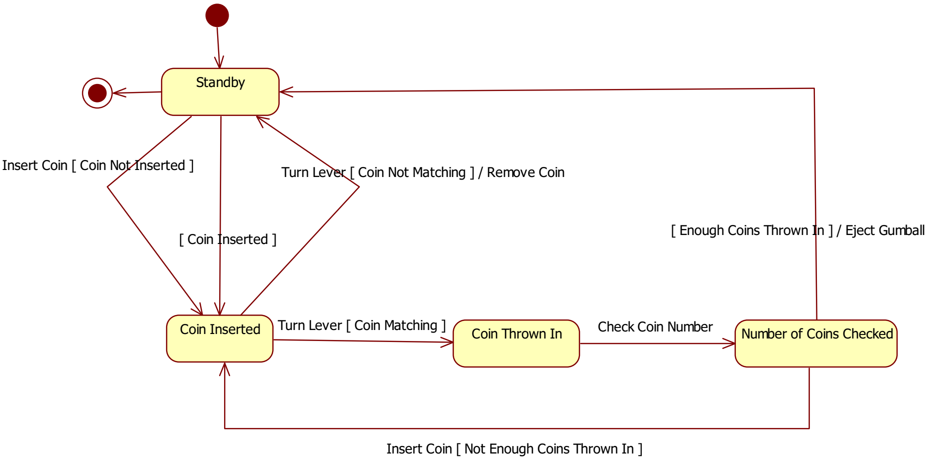
Standby
Insert Coin [Coin Not Inserted]
Turn Lever [Coin Not Matching] / Remove Coin
[Coin Inserted]
[Enough Coins Thrown In] / Eject Gumball
Coin Inserted
Turn Lever [Coin Matching]
Coin Thrown In
Check Coin Number
Number of Coins Checked
Insert Coin [Not Enough Coins Thrown In]

Fig. 3 Test model of gumball machine described as UML state machine used to prepare the participants in the training phase

machine correspond to test steps. The students obtained a sheet for each task with four test case templates on it.

Table 3 shows an example test case derived from the UML activity diagram of the gumball machine shown in Fig. 2.

Table 4 shows an example test case derived from the UML state machine of the gumball machine shown in Fig. 3.

The derived test cases are abstract as they do not contain concrete but abstract input data. We decided to focus on abstract test cases as the derivation of concrete input data would require additional data flow modeling and introduces confusing factors. But already the derivation of abstract test cases, as discussed in our experiment, is of high practical relevance as the creation of test data is in practice often performed independently of the abstract test case derivation and even applies specific test design techniques like equivalence partitioning (Felderer and Beer 2013) which are out of scope of the experiment.

Additionally to the main task of deriving test cases, the participants were also asked several open comprehension questions with respect to the diagrams, in order to verify that they basically understood the respective diagram. Finally, the participants were asked to judge the perceived comprehensibility of the respective diagram on the following five-point Likert-scale: very good = +2, good = +1, average = 0, bad = −1, very bad = −2.

We made available all provided sheets used as experimental material online at http://homepage.uibk.ac.at/~c703409/manual_test_derivation.

We created and internally discussed sample solutions for each of the four sets of test cases. The sets of test cases needed for edge coverage for each of the four diagrams were modeled

Table 3 Example test case derived from the activity diagram in Fig. 2

TC_2	Precondition: no coin inserted		
No.	Operation call	Input data	Expected result
1	Insert coin	Not matching coin	
2	Turn lever		Coin not thrown in
3	Remove coin		No gumball ejected and coin removed

Table 4 Example test case derived from the state machine in Fig. 3

TC_1	Precondition: coin inserted and coin not matching		
No.	Operation call	Input data	Expected result
1	Turn lever		Coin not thrown in
2	Remove coin		Standby
3	Insert coin	Matching coin	
4	Turn lever		Coin thrown in
5	Check coin Number		Not enough coins thrown in
6	Insert coin	Matching coin	
7	Turn lever		Coin thrown in
8	Check coin number		Enough coins thrown in
9	Eject gumball		Gumball ejected and standby

completely. The sample solution contained three test cases for the ATM's activity diagram and four for the DVM's activity diagram. Both state machines demanded only one test case for edge coverage by traversing loops several times. However, semantically they contained the same test cases like the activity diagram's test cases, one executed after the other. We also created sample solutions for all other questions we asked. Each author answered the questions independently and then we compared our solutions and discussed them.

3.3 Experiment execution

The group sizes were 31, 25 and 28 students, respectively (see Table 5). The first two groups were students of Business Informatics at the Duale Hochschule Baden-Württemberg in Karlsruhe in their third year of studies and the experiment took place on the 14th October 2013 during the software engineering lecture. The members of the third group were 28 bachelor students of Computer Science at the University of Innsbruck. This experiment was conducted in a lecture on software quality and took place on the 31st of October 2013. Table 5 shows how many participants from each of the three groups, i.e., Karlsruhe group 1, Karlsruhe group 2, and Innsbruck, have received which of the two treatments, i.e., treatment A and treatment B.

The experiment followed the same agenda for all three groups. As no previous experience with the task of test case derivation could be assumed and we wanted all participants to understand the task and the artifacts used, the experiment was prepared by a previous, similar exercise. Consequently, each experiment included two sessions: the training phase and the actual experiment.

The first session was the training phase. After an explanation of the task, in this session, the students were asked to derive test cases from an activity diagram as well as a state machine describing a gumball machine. The students first received instructions on the task, and then tried themselves to perform the task. The derived test cases are abstract

Table 5 Test design: treatments, groups and group sizes

	Karlsruhe, group 1	Karlsruhe, group 2	Innsbruck	Sum
Treatment A	15	13	14	42
Treatment B	16	12	14	42
Sum	31	25	28	84

as they do not contain concrete but abstract input data. Afterwards, our sample solution was presented and discussed with the whole group of students. This sample solution contained a set of test cases for each of the diagrams which achieves branch coverage and uses the minimum number of loops and steps. The sample solution also included the correct answers to all questions on the questionnaire: the number of test cases needed for each of the three types of coverage and the answers to the comprehensibility questions. The students also received the sample solution. The experiences and feedback from this exercise helped us to make the task description for the experiment more precise, to plan the time schedule and to refine the material used. The observation of the students when deriving test cases and the joint discussion of the solution reflected a broad understanding of the UML diagrams and test derivation from these. Additionally, the two researchers designed sample solutions before executing the experiment, to estimate the required time and the difficulty of the experimental task.

The second session comprised the actual experiment. All students had to perform the same tasks twice with varying domain and diagram type. In both treatments, the DVM model was used first. This was done for the reason if a student asks a question concerning the DVM, it is better that the other group works on the DVM as well to not being confused. During the actual experiment, there were almost no questions asked. The students had as much time as they needed. Their time need was 30 to 70 min for both diagrams. The motivation for the students to participate in the experiment was the hint that a similar task would be part of the course's final exam.

It would have been interesting to log the time need for each individual task of the experiment. However, for several practical reasons, we resigned to log the time. Due to our experience from previous experiments, this would distract the participants from the test case derivation and it is difficult to receive correct and complete time values. In addition, drawing conclusions from the time need would be difficult anyway: a long time could mean slow comprehension, but also high motivation and systematic working style (including re-reading one's results). Therefore, we decided to not log time and to not limit time to avoid any time pressure.

3.4 Analysis procedure

In this section, we describe the analysis procedure applied to answer the six research questions formulated before. According to their formulation, RQ1, RQ2, and RQ3 are answered by hypothesis testing, and RQ4, RQ5, and RQ6 by correlation analysis.

To answer RQ1, the following hypotheses have been tested:

> $H_{1,0}$: There is no significant difference in the number of test step, test case and test suite errors, respectively, made when deriving test cases between UML activity diagrams and state machines.
>
> $H_{1,1}$: Significantly more test step, test case and test suite errors, respectively, are made when deriving test cases from UML activity diagrams than from state machines.

To answer RQ2, the following hypotheses have been tested:

> $H_{2,0}$: There is no significant difference in the self-assessed comprehensibility of UML activity diagrams and state machines.

$H_{2,1}$: UML activity diagrams have a significantly higher self-assessed comprehensibility than UML state machines.

To answer RQ3, the following hypotheses have been tested:

$H_{3,0}$: There is no significant difference in the actual comprehensibility of UML activity diagrams and state machines.

$H_{3,1}$: UML activity diagrams have a significantly higher actual comprehensibility than UML state machines.

The goal of the statistical analysis is to reject the null hypotheses and possibly accept the alternative ones. For all hypotheses, the independent variable is the diagram type (DT), i.e., activity diagram (AD) or state machine (SM). H_1 is evaluated separately for test steps, cases and the overall test suite. The dependent variables are NOE_STEP, NOE_CASE, and NOE_SUITE, respectively. So H_1 in fact covers three hypotheses, i.e., for NOE_STEP, NOE_CASE, and NOE_SUITE. For H_2, the dependent variable is SCOMP of a diagram, and for H_3 ACOMP. We formulate all hypotheses as one-tailed hypotheses because due to personal experiences, initial data exploration, and previous investigations (Felderer and Herrmann 2014) we have specific assumptions about the direction of the cause-effect relationship between the independent and dependent variables: On the one hand, one seems to make more errors when deriving test cases from activity diagrams than from state machines but on the other hand activity diagrams seem to be more understandable than state machines. The aim of the statistical analysis is to reject the null hypotheses and possibly accept the alternative ones. For data measured on an ordinal scale we use non-parametric Wilcoxon tests, for other data we use a Shapiro-Wilk normality test (Crawley 2012), to determine whether the data follows a normal distribution. If Shapiro-Wilk does not refute the assumption that the data is normally distributed, we test by means of a *t* test (Crawley 2012), which is a parametric test, and otherwise, we apply a non-parametric Wilcoxon signed-rank test (Crawley 2012). For all hypotheses, we apply a significance level of $\alpha = 0.05$. If the data is normally distributed, we test H_1 by means of the one-tailed paired *t* test (Crawley 2012), which is a parametric test. Otherwise, we apply the non-parametric one-tailed Wilcoxon signed-rank test (Crawley 2012). For hypothesis H_2, which addresses the difference of the self-assessed comprehensibility between UML activity diagrams (AD) and state machines (SM), we performed—as the dependent variable COMP is measured on an ordinal scale—the non-parametric one-tailed Wilcoxon rank-sum test (Crawley 2012).

For the analysis of correlation between the variables in RQ4, RQ5 and RQ6, we use Spearman's rank correlation coefficient (Crawley 2012) which is appropriate because in every analysis at least one variable is measured on an ordinal scale. The statistical analyses of all research questions are performed within the statistical computing environment R (Crawley 2012).

4 Experimental results

The experiment resulted in 150 sets of test cases which were expected to contain a total of 342 test cases. These contained a total of 1816 errors. Two participants, both in Karlsruhe group 2 and one for each treatment, did not create test cases at all and were therefore excluded from further analysis. In what follows, we present the results and their interpretation with respect to the investigated research questions.

Table 6 Error numbers (NOE_STEP, NOE_CASE, NOE_SUITE) for each of the four diagrams (DVM-AD, SVM-SM, ATM-AD, ATM-SM)

	NOE_STEP	NOE_CASE	NOE_SUITE	Sum
DVM-AD	321	61	41	423
DVM-SM	249	16	47	312
ATM-AD	554	81	32	667
ATM-SM	333	42	39	414
Sum	1457	200	159	1816

(RQ1) The number of errors made when deriving test cases are shown for each of the four diagrams in Table 6.

For the hypotheses H_1 of all three error types, i.e., test steps, cases, and suites, respectively, the *p* value of the Shapiro-Wilk normality tests is lower than 0.05. This finding points to the fact, that the data are not normally distributed, and therefore H_1 was tested according to the analysis procedure with a Wilcoxon signed rank test due to the matched error types. With *p* values of 0.01491, 0.00005 and 0.851, respectively, we can reject the null hypothesis $H_{1,0}$ for test steps and test cases but not for test suites, and conclude that significantly more test step and test case errors are made when deriving test cases from UML activity diagrams than from UML state machines.

(RQ2) The frequencies of self-assessed comprehensibility SCOMP per diagram type are shown in Fig. 4. As mentioned in Section 3.4, hypothesis H_2 was tested with the non-parametric Wilcoxon rank sum test (also known as Mann-Whitney *U* test). With a *p* value of 0.003013, we can reject the null hypothesis $H_{2,0}$ and conclude that UML activity diagrams have a significantly higher self-assessed comprehensibility than UML state machines.

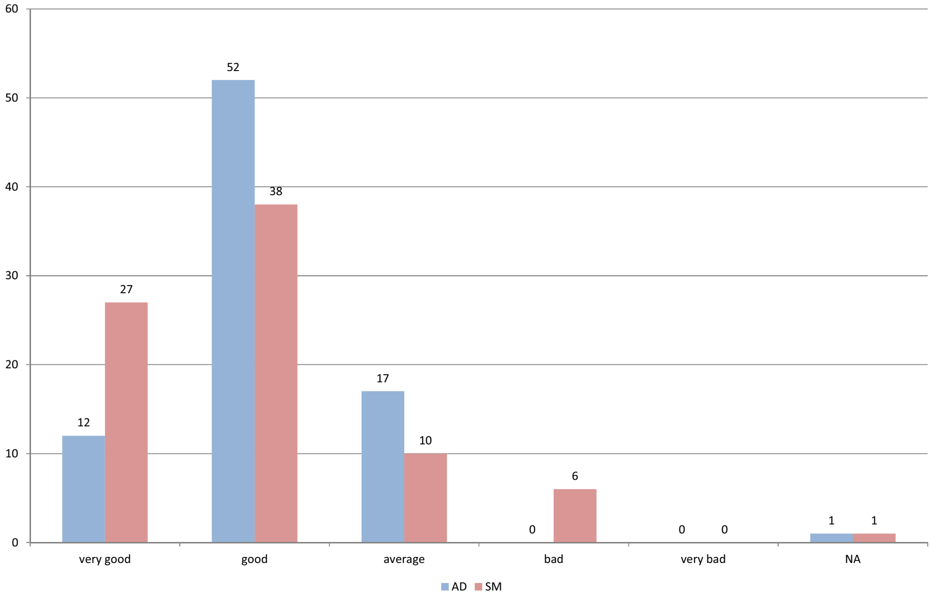

Fig. 4 Frequencies for self-assessed comprehensibility (SCOMP) per diagram type, i.e., activity diagram (AD) and state machine (SM), where NA means value not available

(RQ3) The frequencies of actual comprehensibility ACOMP per diagram type are shown in Fig. 5. As mentioned in Section 3.4, hypothesis H_3 was tested with the non-parametric Wilcoxon rank sum test (also known as Mann-Whitney *U* test). With a *p* value of 0.04076, we can reject the null hypothesis $H_{3,0}$ and conclude that UML activity diagrams have a significantly higher actual comprehensibility than UML state machines.

(RQ4), *(RQ5)*, *(RQ6)* Table 7 shows the Spearman's rank correlation coefficient between error numbers (NOE_STEP, NOE_CASE, NOE_SUITE), self-assessed comprehensibility (SCOMP), as well as actual comprehensibility (ACOMP). Between each pair of two different variables only very weak correlations exist.

5 Discussion

In this experiment, we found that many errors are made when manually deriving system test cases from both UML activity diagrams and state machines. However, there are statistically relevant differences between both diagram types.

Significantly more test step and test case errors are made when deriving test cases from UML activity diagrams than from UML state machines *(RQ1)*. This seems to contradict the finding that activity diagrams were found to be significantly better comprehensible than state machines—both, with respect to the self-assessed comprehensibility *(RQ2)* and the actual comprehensibility *(RQ3)* measured by the correctness of responses to comprehension questions. However, these findings are not really contradicting when considering that model comprehension and test case derivation are two different activities. It can even be considered logical that the activity diagram being less formal is more easily comprehensible, but then

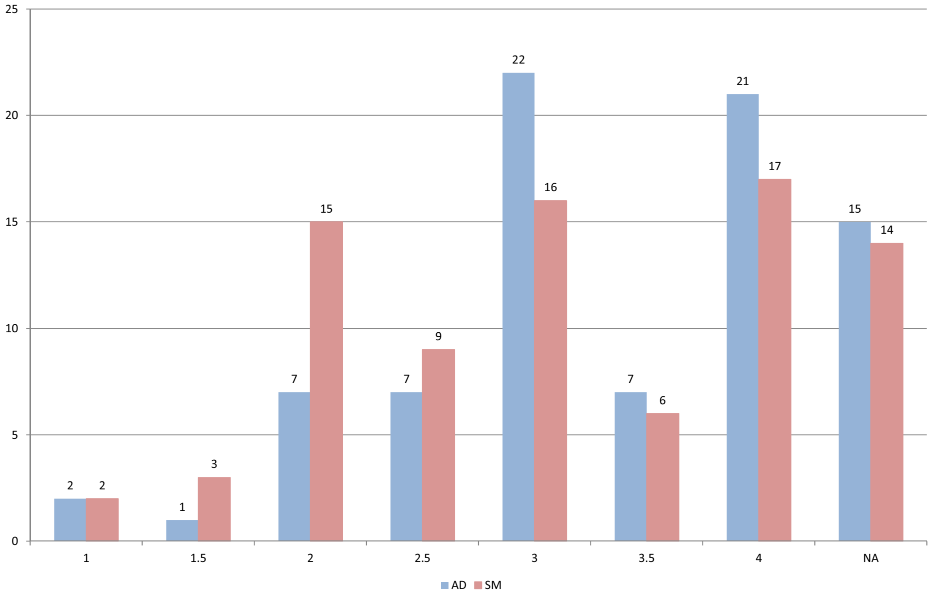

Fig. 5 Frequencies for actual comprehensibility (ACOMP), i.e., sum of points for answering four comprehensibility questions, per diagram type, i.e., activity diagram (AD) and state machine (SM), where NA means sum not available

Table 7 Correlation matrix between error numbers and comprehensibility (SCOMP and ACOMP)

	Activity diagram (AD)		State machine (SM)		
	SCOMP	ACOMP	SCOMP	ACOMP	
NOE_STEP	0.063	0.059	0.057	− 0.049	
NOE_CASE	− 0.069	0.099	0.104	0.069	
NOE_SUITE	− 0.015	0.029	− 0.087	− 0.045	
SCOMP	1.000	0.078	1.000	0.127	RQ6
ACOMP	0.078	1.000	0.129	1.000	
	RQ4	RQ5	RQ4	RQ5	

lacks part of the information needed for test case derivation. For instance, preconditions and expected results are modeled explicitly in the UML state machine as states, whereas in the UML activity diagrams, this information is contained implicitly in the diagram. This shows that the two diagram types might be optimized for different project stakeholders. Activity diagrams might be more easily understandable for customers, while for testers, the more formal state machine fits better because it contains more of the information required for test case derivation.

Additionally to the differences between the two diagram types, we analyzed correlations between the self-assessed comprehensibility and the number of errors made *(RQ4)*, between the actual comprehensibility and the number of errors made *(RQ5)*, and between the self-assessed comprehensibility and the actual comprehensibility *(RQ6)*. These correlation analyses would show when there are correlations between the same person's results: Do those subjects who understood the model well or who thought they understood the model make less or more errors? Do those who thought they understood the model well also give better replies to the comprehensibility questions? However, no such correlations were found. From this, we draw the following conclusions: Model comprehension and test case derivation are different steps and seem to be relatively independent of each other. Understanding the model well does not necessarily lead to better (or worse) test cases. The experiment participants could not well judge their own model comprehension.

As model comprehension alone does not lead to complete test cases, the test designers could profit from guidelines which support them when deriving test cases. Such guidelines can read like: "Each activity in the activity diagram corresponds to one or several test steps.", or: "Each trigger in a state machine corresponds to an operation call of a test step."

Our results are consistent with related work. We found that during manual derivation of test cases from UML models, errors are made. This is similar for other complex activities performed manually during software engineering, like inspections (Runeson et al. 2006). Such activities cannot be performed manually without errors. Fully automated test derivation may help to overcome this issue. However, this requires complete and correct test models, which need a great effort and respective expertise to be created and maintained. Therefore, manual test derivation is still of use and can help to detect and correct defects in the UML model and therefore serves as a quality assurance activity.

Deriving test cases manually is error-prone and requires a significant manual effort. This provides a motivation to automate the derivation of test cases from UML models. Approaches for doing so exist. Some authors use state machines to derive test cases (e.g., (Kim et al. 1999; Samuel et al. 2008; Kansomkeat and Rivepiboon 2003)), others use activity diagrams (Linzhang et al. 2004; Mingsong et al. 2009; Kim et al. 2007; Tripathy and Mitra 2013), and still others both (Swaina et al. 2010). However, many authors highlight that the UML models without additional annotation do not contain sufficient information for automated test generation. Therefore, state diagrams are for instance enhanced by OCL expressions (Offutt and Abdurazik 1999; Weißleder and Sokenou 2008), activity diagrams are enhanced by OCL expressions and further information (Briand and Labiche 2002b; Hartmann et al. 2005). Other researchers combine two types of UML models, e.g., state machines with activity diagrams (Swaina et al. 2010) or activity diagrams with sequence diagrams (Tripathy and Mitra 2013). In practice, this means that the preparation for the automated test case derivation (similar to the manual derivation that we discuss in this paper) also requires time-consuming and potentially error-prone manual preparations.

In the literature, we found no comparison of the comprehensibility and error-proneness of UML activity diagrams and state machines with regard to manual test case derivation. So this paper is the first research contribution showing the easier comprehensibility and higher error-proneness of activity diagrams compared to state machines when manually deriving test cases.

6 Threats to validity

In this section, we discuss the validity of the experiment, i.e., its internal, external, construction, and conclusion validity (Wohlin et al. 2012).

Internal validity is threatened if a relationship is observed between the treatment and the outcome, although there in fact is none. This can happen when the observed effect is caused by the treatment. However, we took care that the two diagrams presenting the same machine were semantically equal. Another potential threat is the exchange of information among the participants. We emphasize that participants were not allowed to communicate with each other; we prevented this arranging them appropriately, i.e., students with the same treatment did not sit next to each other, and we monitored them during the experiment.

In our specific experiment, the errors made by the participants depend strongly on the quality of the explanations and instructions they obtained. We executed the experiment with three groups who received the same explanations. They were prepared by a previous exercise, so there was a feedback loop for the participants and also for the experimenters. We understand our experiment results as a hint on where the training process can be improved (e.g., by providing explicit guidelines), so the participants would make fewer errors. We do not expect that there are differences between the Business Informatics students and the Computer Science students because their previous knowledge with respect to the task to execute was similar. All students knew UML models from previous courses, received the same introductory training and were familiar with the chosen domains. In addition, neither specific business nor technical knowledge is required to perform the experiment task. We tested our assumption that there are no differences between the three groups by a Kruskal-Wallis one-way analysis of variance (Crawley 2012), which confirmed that the perceived comprehensibility and number of errors made for activity diagrams and state machines do not differ significantly between the three groups.

Construct validity refers to the extent to which the experiment setting or the measure chosen actually reflects the construct under study. The metrics to answer the research questions were defined based on a literature research about which metrics are commonly used. Students were given enough time to answer all questions and to perform the experimental tasks. Social threats (e.g., evaluation apprehension) have been avoided, since the students were not graded on the results obtained.

When comparing to comprehensibility measured by multiple choice questions, the F-measure from the multiple choice questions is mathematically comparable to the ratio of correct free text answers (Gravino et al. 2008). The free text questions in our experiment could either be correct (1 point), incorrect (0), or partly correct (1/2 point). Our participants achieved a ratio of 0.66 correct answers for the activity diagram of the DVM and 0.64 for the ATM activity diagram. This is comparable to the results from a similar series of student experiments (Reggio et al. 2011) where an F-measure of 0.52 to 0.64 was achieved for activity diagrams. The state machines achieved 0.56 correct answers for the DVM and 0.74 for the ATM. So, our participants' model comprehension was in a good range.

Conclusion validity is concerned with data collection, the reliability of the measurement, and the relationship of the treatment and the outcome, i.e., whether there is a statistically significant relationship. In our experiment, 84 students participated. This is a sufficient sample size providing enough data to derive significant results.

External validity is concerned with the extent to which it is possible to generalize the findings. It is an important issue in student experiments. The results of our experiment cannot be generalized to all industrial settings and to professional test designers. Experienced testers would probably derive more complete test cases. Furthermore, our results are not generalizable to systems of arbitrary complexity, due to the rather low complexity of the considered problems and the simplicity of the provided diagrams. Both limitations result from the experiment's time constraints. But also in industry, a remarkable amount of the diagrams are only simple and illustrative, and testers not always are testing experts but rather domain experts with only little experience in systematic testing (Felderer et al. 2014). The required skills in test design are then provided in short trainings (i.e., similar to the training phase in our experiment). This increases the transferability of our results to industrial settings.

We use this experiment for gaining insights into what errors are intuitively made by subjects who have low previous experience in deriving test cases, but who have been trained before. To test whether experienced testers make fewer errors is considered as future work.

7 Conclusions and future work

In this paper, we empirically evaluated the comprehensibility of UML activity diagrams and state machines during manual derivation of test cases in a controlled experiment and a replication with student participants. UML activity diagrams were found to be statistically significantly more comprehensible than UML state machines, according to the experiment subjects' self-assessment as well as measured by correct replies to comprehension questions. However, more errors were made when deriving test cases from UML activity diagrams than from UML state machines. When analyzing correlations on an individual level, model comprehensibility and errors made during test case derivation were not found to correlate. And the two measures of comprehensibility were also not correlated, which seems to imply that participants could not quantify their own model's comprehension.

We will continue following this direction of research. Two types of activities are planned: First, we will analyze the data generated from the experiment further and investigate additional research questions and hypotheses. For instance, we also asked questions which investigate the personal characteristics of the participants and will try to find correlations between personality and errors made. If there are correlations, this means that a good tester can be chosen based on a personality test. The other line of future research is to execute further controlled experiments. For instance, we plan to give the subjects more detailed guidelines and then to investigate whether these improvements indeed reduce not only the total number of errors, but also which concrete errors types are actually reduced. As the manual derivation of test cases from activity diagrams or state charts is also common in practice, we plan industrial case studies to investigate these research questions and to apply guidelines for manual test derivation in an industrial context.

Acknowledgements Open access funding provided by University of Innsbruck and Medical University of Innsbruck. We thank all participants of the experiment for their time and concentration.

Funding information This work was partly funded by the Austria Science Fund (FWF) through the project MOBSTECO (FWF P 26194-N15) as well as by the Knowledge Foundation (KKS) of Sweden through the project 20130085: Testing of Critical System Characteristics (TOCSYC).

References

Agarwal, R., De, P., & Sinha, A. P. (1999). Comprehending object and process models: an empirical study. *IEEE Transactions on Software Engineering, 25*(4), 541–556.

Aranda, J., Ernst, N., Horkoff, J., Easterbrook, S. (2007) A framework for empirical evaluation of model comprehensibility. International workshop on modeling in software engineering (MiSE-07) .

Briand, L., & Labiche, Y. (2002a). A UML-based approach to system testing. *Software and Systems Modeling, 1*(1), 10–42.

Briand, L., & Labiche, Y. (2002b). A UML-based approach to system testing. *Software and Systems Modeling, 1*(1), 10–42.

Briand, L. C., Labiche, Y., Di Penta, M., & Yan-Bondoc, H. (2005). An experimental investigation of formality in UML-based development. *IEEE Transaction on Software Engineering, 31*(10), 833–849.

Budgen, D., Burn, A. J., Brereton, O. P., Kitchenham, B. A., & Pretorius, R. (2011). Empirical evidence about the UML: a systematic literature review. *Software: Practice and Experience, 41*(4), 363–392.

Cioch, F. A. (1991). Measuring software misinterpretation. *Journal of Systems and Software, 14*(2), 85–95.

Crawley, M. J.(2012) The R book. Wiley.

Cruz-Lemus, J. A., Genero, M., Manso, M. E., Piattini, M. (2005). Evaluating the effect of composite states on the understandability of UML statechart diagrams. in: Model driven engineering languages and systems (MoDELS 2005), 113–125.

Cruz-Lemus, J. A., Genero, M., Caivano, D., Abrahão, S., Insfrán, E., & Carsí, J. A. (2011a). Assessing the influence of stereotypes on the comprehension of UML sequence diagrams: a family of experiments. *Information and Software Technology, 53*(12), 1391–1403.

Cruz-Lemus, J. A., Genero, M., Caivano, D., Abrahão, S., Insfrán, E., & Carsí, J. A. (2011b). Assessing the influence of stereotypes on the comprehension of UML sequence diagrams: a family of experiments. *Information and Software Technology, 53*(12), 1391–1403.

De Lucia, A., Gravino, C., Oliveto, R., & Tortora, G. (2010). An experimental comparison of ER and UML class diagrams for data modelling. *Empirical Software Engineering, 15*(5), 455–492.

Eichelberger, H., & Schmid, K. (2009). Guidelines on the aesthetic quality of UML class diagrams. *Information and Software Technology, 51*(12), 1686–1698.

Felderer, M., & Beer, A. (2013). Using defect taxonomies to improve the maturity of the system test process: results from an industrial case study. *In Software Quality. Increasing Value in Software and Systems Development*.

Felderer, M., & Herrmann, A. (2014). Manual test case derivation from UML activity diagrams and state machines: a controlled experiment. *Information and Software Technology, 61*, 1–15.

Felderer, M., Beer, A., Peischl B. (2014). On the role of defect taxonomy types for testing requirements: results of a controlled experiment, Euromicro SEAA 2014.

Fernández-Sáez, A. M., Genero, M., & Chaudron, M. R. V. (2013). Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: a systematic mapping study. *Information and Software Technology, 55*(7), 1119–1142.

Genero, M., Cruz-Lemus, J. A., Caivano, D., Abrahão, S., Insfran, E., & Carsí, J. A. (2008). Assessing the influence of stereotypes on the comprehension of UML sequence diagrams: a controlled experiment. *MoDELS, 2008*, 280–294.

Glezer, A., Last, M., Nachmany, E., & Shoval, P. (2005). Quality and comprehension of UML interaction diagrams-an experimental comparison. *Information and Software Technology, 47*(10), 675–692.

Granda, F. M., Condori-Fernández, N., Vos, T., & Pastor, O. (2014). Towards the automated generation of abstract test cases from requirements models. *RET Workshop*.

Gravino, C., Scanniello, G., & Tortora, G. (2008). An empirical investigation on dynamic modeling in requirements engineering. *MoDELS, 2008*, 615–629.

Hartmann, J., Vieira, M., Foster, H., & Ruder, A. (2005). A UML-based approach to system testing. *Innovations System Software Engineering, 1*(1), 12–24.

ISTQB. (2012). Standard glossary of terms used in software testing. *Version, 2*, 2.

Juristo, N., Moreno, A. M., & Vegas, S. (2004). Reviewing 25 years of testing technique experiments. *Empirical Software Engineering, 9*(1–2), 7–44.

Kansomkeat, S., & Rivepiboon, W. (2003). Automated-generating test case using UML statechart diagrams. In *SAICSIT '03*.

Kim, Y. G., Hong, H. S., Bae, D.-H., & Cha, S. D. (1999). Test cases generation from UML state machines. *IEEE Software, 146*(4), 187–192.

Kim, H., Kang, S., Baik, J., & Ko, I. (2007). Test cases generation from UML activity diagrams. *In Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/ Distributed Computing*.

Kirk, R.E.(1995) Experimental design. Procedures for the behavioural sciences. Brooks/Cole Publishing Company.

Kundu, B., & Samanta, D. (2009). A novel approach to generate test cases from UML activity diagrams. *Journal of Object Technology, 8*(3), 65–83.

Lindland, O., Sindre, G., Sølvberg, A. (1994) Understanding quality in conceptual modeling, 42–49.

Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L., Guoliang, Z.(2004) Generating test cases from UML activity diagram based on gray-box method. 11th Asia-Pacific Software Engineering Conference.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 2*(4), 308–320.

Mendonça, M. G., Maldonado, J. C., de Oliveira, M. C. F., Carver, J., Fabbri, S. C. P. F., Shull, F., Travassos, G. H., Höhn, E. N., & Basili, V. R. (2008). A framework for software engineering experimental replications. *13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008)*, 203–212.

Mingsong, C., Xiaokang, Q., Xuandong, L.(2006) Automatic test case generation for UML activity diagrams. Proceedings of the 2006 international workshop on Automation of software test (AST 06).

Mingsong, C., Xioakang, Q., Wei, X., Linzhang, W., Jianhua, Z., & Xuandong, L. (2009). UML activity diagram-based automatic test case generation for Java programs. *The Computer Journal, 52*(5), 545–556.

Mohacsi, M., Felderer, M., & Beer, A. (2015). Estimating the cost and benefit of model-based testing: a decision support procedure for the application of model-based testing in industry. *Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2015)*, 382–389.

Nugroho, A. (2009). Level of detail in UML models and its impact on model comprehension: a controlled experiment. *Information and Software Technology, 51*(12), 1670–1685.

Nugroho, A., & Chaudron, M. R. V. (2009). Evaluating the impact of UML modeling on software quality: an industrial case study. *MoDELS*, 181–195, 2009.

Nugroho, A., & Chaudron, M. R. V. (2014). The impact of UML modeling on defect density and defect resolution time in a proprietary system. *Empirical Software Engineering, 19*(4), 926–954.

Offutt, J., Abdurazik, A.(1999) Generating tests from UML specifications. UML99, LNCS 1723.

Otero, M. C., & Dolado, J. J. (2004). Evaluation of the comprehension of the dynamic modeling in UML. *Information and Software Technology, 46*(1), 35–53.

Pohl, K., Rupp, C. (2011). Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam-foundation level-IREB compliant. O'Reilly.

Pretschner, A., Prenninger, W., Wagner, S., Kühnel, C., Baumgartner, M., Sostawa, B., Zölch, R., Stauner, T.(2005) One evaluation of model-based testing and its automation. In Proceedings of the 27th international conference on software engineering. ACM.

Purchase, H. C., Colpoys, L., McGill, M., Carrington, D., & Britton, C. (2001). UML class diagram syntax: an empirical study of comprehension. In *Proceedings of the 2001 Asia-Pacific symposium on information visualisation-volume 9*.

Reggio, G., Ricca, F., Scanniello, G., Di Cerbo, F., & Dodero, G. (2011). A precise style for business process modelling: results from two controlled experiments. Model driven engineering languages and systems. *Springer Berlin Heidelberg*, 138–152.

Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., & Ceccato, M. (2010). How developers' experience and ability influence web application comprehension tasks supported by UML stereotypes: a series of four experiments. *IEEE Transaction on Software Engineering, 36*(1), 96–118.

Riebisch, M., Philippow, I., Götze, M.(2003) UML-based statistical test case generation. Objects, Components, Architectures, Services, and Applications for a Networked World. LNCS 2591.

Runeson, P., Andersson, C., Thelin, T., Andrews, A., & Berling, T. (2006). What do we know about defect detection methods? *IEEE Software, 23*(3), 82–90.

Samuel, P., Mall, R., & Bothra, A. K. (2008). Automatic test case generation using unified modeling language (UML) state machines. *IET Software, 2*(2), 79.

Sharif, B., & Maletic, J. (2009). An empirical study on the comprehension of stereotyped UML class diagram layouts. *17th International Conference on Program Comprehension*, 268–272.

Sharif, B., & Maletic, J. (2010). An eye tracking study on the effects of layout in understanding the role of design patterns. *IEEE International Conference on Software Maintenance (ICSM)*, 1–10.

Staron, M., Kuzniarz, L., & Wohlin, C. (2006). Empirical assessment of using stereotypes to improve comprehension of UML models: a set of experiments. *Journal of Systems and Software, 79*(5), 727–742.

Störrle, H. (2012). On the impact of layout quality to understanding UML diagrams: diagram type and expertise. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HC 2012)*, 49–56.

Störrle, H. (2014). On the impact of layout quality to understanding UML diagrams: size matters. *Model-Driven Engineering Languages and Systems (MoDELS 2014)*, 518–534.

Swaina, S. K., Mohapatrab, D. P., & Mallc, R. (2010). Test case generation based on state and activity models. *Journal of Object Technology, 9*(5), 1–27.

Tripathy, A., Mitra, A.(2013) Test case generation using activity diagram and sequence diagram. Kumar A. et al. (Eds.): Proceedings of ICAdC, AISC 174, 121–129.

Utting, M., Pretschner, A., & Legeard, B. (2012). A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability, 22*(5), 297–312.

Weißleder, S., & Sokenou, D. (2008). Automatic test case generation from UML models and OCL expressions. *Software Engineering (Workshops)*.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. *Spring*.

Wong, K., & Sun, D. (2006). On evaluating the layout of UML diagrams for program comprehension. *Software Quality Journal, 14*(3), 233–259.

Michael Felderer is a professor in software engineering at the Institute of Computer Science at the University of Innsbruck, Austria, and a guest professor at the Blekinge Institute of Technology, Sweden. He holds a PhD and a habilitation degree in computer science. His research interests include software testing and software quality in general, risk management, requirements engineering, model engineering, empirical software engineering, software processes, security engineering, software analytics, and improving industry-academia collaboration. He works in close collaboration with industry and transfers his research results into practice as a consultant and speaker on industrial conferences.

Andrea Herrmann is a freelance trainer and consultant in software engineering with 20 years of work experience. She holds a PhD in Physics and a habilitation degree in computer science. Her research interests include requirements engineering, empirical software engineering, and IT project management. She wrote more than 100 publications, regularly speaks at conferences, and is an associate member at IREB, co-author of syllabus and handbook for the CPRE Advanced Level in Requirements Management, and speaker of the regional group of the German Informatics Society in Stuttgart.