Empirical Evaluation of Test Driven Modeling

Stefan Zugal, Cornelia Haisjackl, Jakob Pinggera and Barbara Weber
University of Innsbruck, Austria

ABSTRACT

Declarative approaches to process modeling are regarded well suited for highly volatile environments as they provide a high degree of flexibility. However, problems in understanding and maintaining declarative process models impede their usage. To compensate for these shortcomings, Test Driven Modeling (TDM) has been proposed. This paper reports on an empirical investigation in which TDM is viewed from two different angles. First, the impact of TDM on communication is explored in a case study. Results indicate that domain experts are inclined to use test cases for communicating with the model builder (system analyst) and prefer them over the process model. The second part of the investigation, a controlled experiment, investigates the impact of TDM on process model maintenance. Data gathered in this experiment indicates that the adoption of test cases significantly lowers cognitive load and increases the perceived quality of changes.

Keywords: Business Process Management, Declarative Business Process Model, Test Driven Modeling, Empirical Research, Case Study, Controlled Experiment.

INTRODUCTION

In today's dynamic business environment, the economic success of an enterprise depends on its ability to react to various changes like shifts in customer's attitudes or the introduction of new regulations (Lenz & Reichert, 2007). Process-Aware Information Systems (PAISs) offer a promising perspective on shaping this capability, resulting in growing interest to align information systems in a process-oriented way (Dumas, van der Aalst, & ter Hofstede, 2005). Yet, a critical success factor in applying PAISs is the possibility of flexibly dealing with process changes (Lenz & Reichert, 2007). To address the need for flexible PAISs, competing paradigms enabling process changes and process flexibility have been developed, e.g., adaptive processes (Reichert & Dadam, 1998), case handling (van der Aalst W. , 2005) or declarative processes (Pesic, Schonenberg, Sidorova, & van der Aalst, 2007); an overview is provided in (Weber, Reichert, & Rinderle, 2008).
Especially declarative processes have recently attracted the interest of researchers, as they provide a high degree of flexibility (Weber, Reichert, & Rinderle, 2008). Nevertheless, declarative processes are not widely adopted in practice yet. In particular, as pointed out in (Pesic, 2008; Zugal, Pinggera, & Weber, 2012), understandability problems and maintainability problems hamper the usage of declarative process models (for a general discussion about model understandability, we refer to (Zugal, Pinggera, & Weber, 2011a; Zugal S. , Pinggera, Weber, Mendling, & Reijers, 2011)). An approach tackling these problems, the Test Driven Modeling (TDM) methodology, is presented in (Zugal, Pinggera, & Weber, 2012). TDM aims at improving the understandability and maintainability of declarative process models as well as the communication between domain expert and model builder

(system analyst) by adopting the concept of test cases from software engineering. While the proposed concepts seem to be beneficial from a theoretical point of view, no comprehensive empirical evaluation exists yet. The goal of this paper is to extend previous work (Zugal, Pinggera, & Weber, 2011c) and provide a more comprehensive empirical evaluation. Hence, in this paper, we will provide two complementary perspectives. First, we will embrace a qualitative angle and report the results of a case study. Therein, we focus on the question in how far communication between the domain expert and the model builder is influenced by the adoption of TDM. Second, we will look into the impact of TDM on the maintainability of declarative process models in a controlled experiment, i.e., adopting a quantitative angle. The contribution of this paper is twofold. On the one hand, we report from a case study that investigates the impact of TDM on communication. On the other hand, results from a controlled experiment are integrated to provide a more comprehensive picture.

The remainder of this paper is structured as follows. First, background information about TDM is provided. Then, results from a case study and a controlled experiment are reported. Insights and limitations are subsequently discussed before the paper is closed with the conclusion.

BACKGROUND

In this section background information is provided. First, declarative processes and associated problems are discussed. Then, it is sketched how TDM aims at resolving these problems.

Declarative Processes

There has been a long tradition of modeling business processes in an imperative way. Process modeling languages supporting this paradigm, like BPMN, EPC and UML Activity Diagrams, are widely used. Recently, declarative approaches have received increasing interest and suggest a fundamentally different way of describing business processes (Pesic, 2008). While imperative models specify exactly how things have to be done, declarative approaches only focus on the logic that governs the interplay of actions in the process by describing the activities that can be performed, as well as constraints prohibiting undesired behavior. An example of a constraint in an aviation process would be that crew duty times cannot exceed a predefined threshold. Constraints described in literature can be classified as *execution* and *termination* constraints (Zugal, Pinggera, & Weber, 2012). Execution constraints, on the one hand, restrict the execution of activities, e.g., an activity can be executed at most once. Termination constraints, on the other hand, affect the termination of process instances and specify when process termination is possible. For instance, an activity must be executed at least once before the process can be terminated. Most constraints focus either on execution or termination semantics. However, some constraints also combine execution and termination semantics (e.g., the succession constraint (Pesic, 2008)).

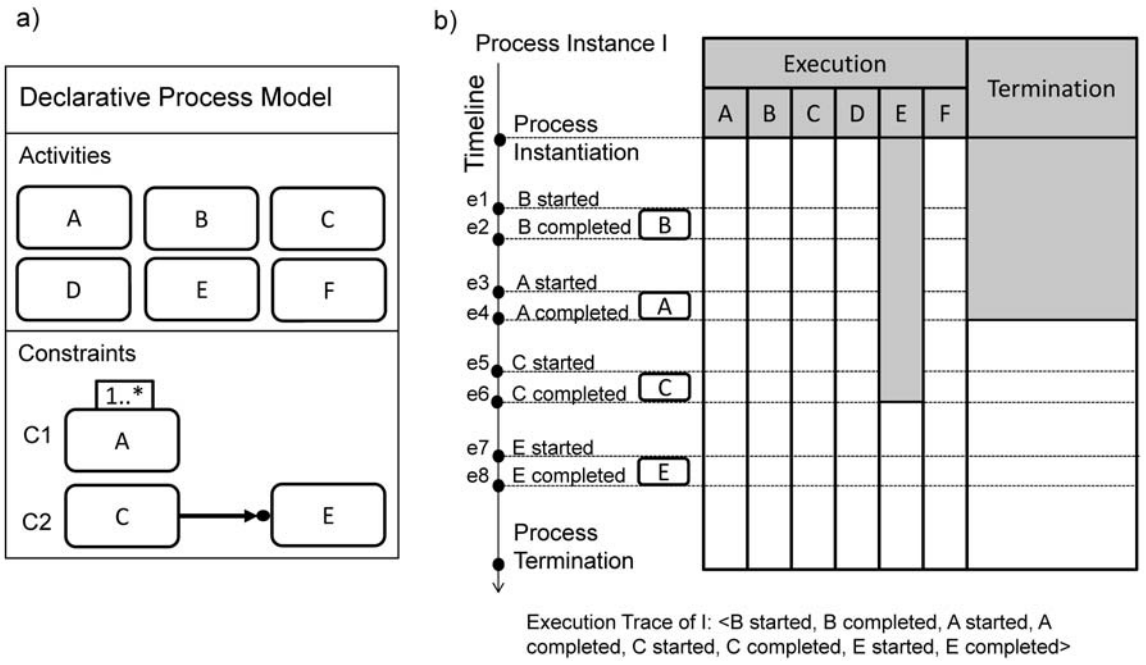


Figure 1. Executing a declarative process

To illustrate the concept of declarative processes, a declarative process model is shown in Figure 1 a). It contains activities *A* to *F* as well as constraints *C1* and *C2*. *C1* prescribes that A must be executed at least once (i.e., *C1* restricts the termination of process instances). *C2* specifies that *E* can only be executed if *C* has been executed at some point in time before (i.e., *C2* imposes restrictions on the execution of activity *E*). In Figure 1 b) an example of a process instance illustrates the semantics of the described constraints. After process instantiation, *A*, *B*, *C*, *D* and *F* can be executed. *E*, however, cannot be executed as *C2* specifies that *C* must have been executed before (cf. grey bar in Figure 1 b) below "*E*"). Furthermore, the process instance cannot be terminated, as *C1* is not satisfied. After *A* has been completed (*e4*), *C1* is satisfied, i.e., the process instance may be terminated. After *C* has been executed (*e6*), *C2* is satisfied, i.e., *E* may be executed.

As illustrated in Figure 1, a process instance can be specified through a list of events that describe changes in the life-cycle of activity instances, e.g., *"e1: B started"*. In the following, we will denote this list as execution trace, e.g., for process instance *I*: <*e1*, *e2*, *e3*, *e4*, *e5*, *e6*, *e7*, *e8* >. If activities are non-overlapping, we merge subsequent start events and events, e.g., <*B started*, *B completed*, *A started*, *A completed*> is abbreviated by <*B*, *A*>.

While the declarative way of process modeling allows for a high degree of flexibility, this freedom comes at the cost of understandability problems (Zugal, Pinggera, & Weber, 2012) and maintainability problems (Weber, Reijers, Zugal, & Wild, 2009). In particular, declarative process models are hard to read and understand, since the interactions between constraints quickly become too complex for humans to deal with (Pesic, 2008). In the following, we will substantiate this claim by two insights from cognitive psychology. First, the *mental execution* of a process instance, i.e., simulating a process instance in one's mind, is not trivial. As discussed, the reader has to interpret constraints and keep track of the execution trace. In imperative process modeling languages, such simulations are supported by *computational offloading* (Zugal, Pinggera, & Weber, 2011a). In other words, the diagram allows extracting this information easily. For declarative process models, however, this support is missing. Therefore, checking whether an execution trace is supported by a process model presumably imposes a high *cognitive load*. The second insight is provided by the concept of *hidden dependencies*, i.e., interactions that are not easily recognizable (Green & Petre, 1996). It is known from Visual Programming Languages that such dependencies hamper understandability. In the following, we will show that hidden dependencies are also present in declarative process models. Consider, for instance, the combination of a cardinality constraint (i.e., an activity must be executed a specific number of times) and a precedence constraint (i.e., an activity must be preceded by another activity) as illustrated in Figure 2. Activity *B* has a cardinality of 1 (i.e., must be executed exactly once) and activity *A* is a prerequisite of *B*. Hence, in order to fulfill both constraints, *A* must be executed at least once. Since this interaction is not explicitly visible, it is not sufficient that the modeler only relies on the information that is displayed explicitly, but has to carefully examine the process model for these hidden dependencies.

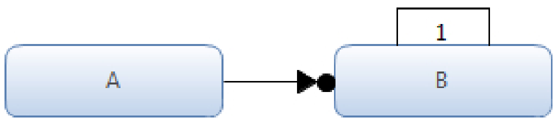


Figure 2. Hidden dependency

Regarding model maintenance, e.g., due to changes in the real-world business process, these limitations compromise adaptions in two ways: In general, as described in (Green & Petre, 1996), any change operation can be broken down into sense-making tasks, i.e., determining what to change and action tasks, i.e., perform the change. Declarative process models, as discussed, exhibit understandability issues that impede the sense-making task. This in turn hampers the action tasks and thus compromises the maintainability of declarative process models. More specifically, *hidden dependencies* make it harder to see what parts of the model are affected by an adaptation—more informally described as *” it is notoriously difficult to determine which constraints have to be modified”* (Weber, Reijers, Zugal, & Wild, 2009). In addition, *lack of computational offloading increases the cognitive load* which, in turn, increases the likelihood of errors (Miller, 1956).

Test Driven Modeling

So far we discussed the benefits and drawbacks of declarative process models, now we briefly sketch how TDM is intended to target these problems (for a detailed discussion we refer to (Zugal, Pinggera, & Weber, 2012)). A central aspect of TDM are so-called *test cases*, which allow specifying behavior the process model must exhibit or prohibit. As the focus of TDM is put on control flow aspects[1], test cases provide mechanisms for the validation of control-flow related properties.
In particular, a test case consists of an execution trace (i.e., the current state of a process instance) as well as a set of assertions (i.e., conditions that must hold for a process instance being in a certain state). The execution trace thereby specifies behavior that must be supported by the process model, whereas assertions allow to test for unwanted behavior, i.e., behavior that must be prohibited by the process

[1] Other perspectives, such as data and resources, are not taken into account by TDM yet.

model. A typical example for an assertion would be to check, whether or not activity *N* is executable after event *e*.
Consider, for illustration, the test case depicted in Figure 3. It contains the execution trace <*A*, *B*> (1) as well as an assertion that specifies that *A* cannot be executed between *e2* and *e3* (2) and assertions that specify that the process instance cannot be terminated before *e2* (3), however, it must be possible to terminate after *e2* (4).

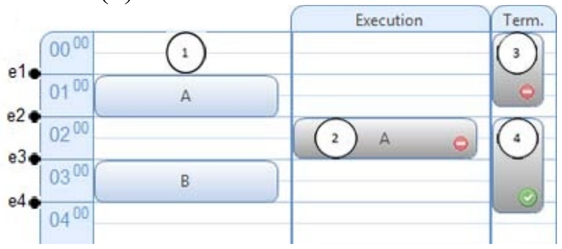


Figure 3. A test case

As illustrated in Figure 3, test cases make information explicit that is only available in an implicit form in process models. For instance, in this example the process instance cannot be terminated until *A* has been executed, cf. termination assertion (3) and (4). Thus, test cases provide an additional view on the process model, which allows resolving *hidden dependencies* by specifying test cases that make these dependencies explicit. In addition, test cases allow testing for execution traces in a computer-based way. Similar to unit testing (Beck, 2002), test cases can be validated *automatically* by replaying the execution trace in a test environment and checking the assertions step-by-step. Hence, test cases compensate for the lacking *computational offloading*: The computer takes over this task, presumably *lowering cognitive load*.

With respect to maintenance, the close coupling of test cases and process model should help to ensure that changes conducted to the process model do not violate desired behavior (cf. regression testing in software engineering (Agrawal, Horgan, Krauser, & London, 1993)). Since test cases are checked automatically, undesired changes to the process model, as caused by *hidden dependencies*, are presumably prevented. As the validation of test cases is computerized, a *lower cognitive load* during model maintenance can be expected. Indeed, it is known from software engineering that test cases are able to *improve perceived quality* (Marchenko, Pekka, & Ihme, 2009) and to improve *quality* (George & Williams, 2004).

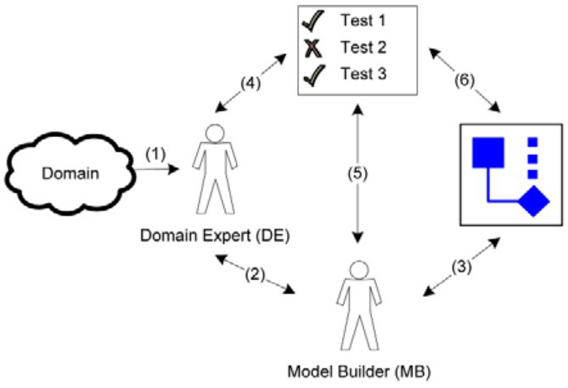
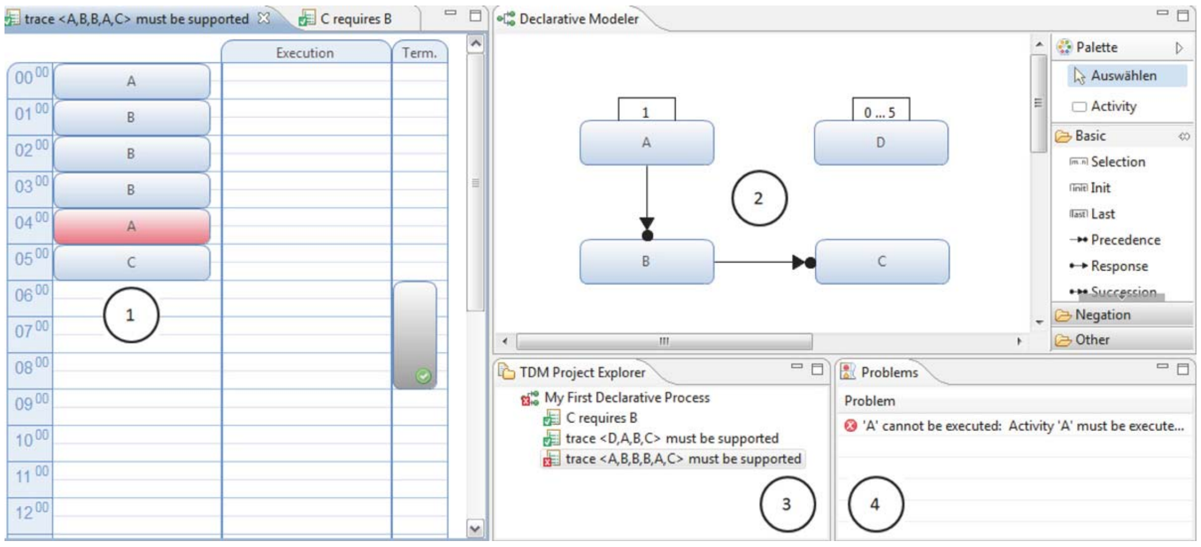


Figure 4. Communication model

So far we have introduced the concept of test cases and the intended impact on model understandability and maintainability. In the following, we will sketch how the adoption of test cases intends to improve the communication during model creation. Thereby, we differentiate between domain expert (DE) and model builder (MB) (Hoppenbrouwers, Lindemann, & Proper, 2006): the DE is responsible for providing information about the domain to be modeled, cf. Figure 4 (1), whereas the MB then converts this information into a formal process model, cf. Figure 4 (3). Please note that we talking here about the phase of process *modeling*, but not about process *execution*, i.e., roles DE and MB refer to the role that *models* an activity rather than *executes* the activity.



'A' cannot be executed: Activity 'A' must be execute...

Figure 5. Test Driven Modeling Suite

Test cases provide information in a form that is not only understandable to the MB, but presumably also understandable to the DE, who normally does not have the knowledge to read *formal* process models (van Bommel, Hoppenbrouwers, Proper, & van der Weide, 2006).

As inspired by Test Driven Development (Beck, 2002), test cases and process model should be created interwoven, i.e., at the same time. Usually the DE needs the MB to retrieve information from the model, cf. Figure 4 (2) and (3). Since test cases are understandable to the DE, they provide an additional communication channel to the process model; cf. Figure 4 (4) and (6). It is important to stress that TDM's intention is not to make the DE specify the test cases in isolation. Rather, test cases should be created by the DE and the MB together and provide a common basis for discussion.

The concepts of TDM are implemented by Test Driven Modeling Suite (TDMS)[2]. As detailed in (Zugal, Pinggera, & Weber, 2011b), TDMS provides a modeling environment for the creation of test cases and process model. In particular, as shown in Figure 5, in TDMS the editors for creating test cases (1) and process model (2) are positioned side-by-side. Whenever a test case or the process model is changed, test cases are validated immediately; possibly failed test cases are indicated in the test case editor (1) and in the test case overview (3). In the problems view, a detailed error message is displayed (4). Any relevant user interaction is immediately logged, allowing for a detailed step-by-step analysis of modeling sessions (Zugal, Pinggera, & Weber, 2011b). Please note that this work focuses on the empirical evaluation of TDM, hence and due to space restrictions, TDM and TDMS are only briefly explained. For more detailed discussions, we refer to (Zugal, Pinggera, & Weber, 2012; Zugal, Pinggera, & Weber, 2011b)

RESEARCH QUESTIONS

In the following, we will empirically investigate the claims of TDM, i.e., the impact on communication behavior and maintenance. In particular, we will start with the definition of the research questions. Research questions 1 to 4 (RQ 1 to RQ 4) take a qualitative viewpoint to investigate the adoption of TDM in a case study. Then,

[2] Freely available from: www.zugal.info/tdms

research questions 5 to 7 (RQ 5 to RQ 7) assess the impact of test cases on the maintainability of *declarative* process models in a controlled experiment. We would like to stress here that the research questions focus on *declarative* process models, as TDM was developed to support the creation and maintenance of *declarative* process models only. Accordingly, we formulated the following research questions.

Research Question 1 (RQ 1): TDM assumes that test cases provide an additional communication channel between DE and MB. The goal of RQ 1 is to investigate whether test cases are actually used for communication.
Research Question 2 (RQ 2): In TDM, process model and test cases are available during modeling. TDM assumes that a DE normally does not have the ability to read formal process models. However, TDM claims that test cases are understandable to the DE and hence likelier to be used for communication than the process model. The goal of RQ 2 is to find out whether test cases are indeed favored over the process model, as suggested by TDM.
Research Question 3 (RQ 3): TDM allegedly fosters communication by providing a common basis for discussion via test cases. The goal of RQ 3 is to investigate whether the adoption TDM positively influences the communication behavior.
Research Question 4 (RQ 4): TDM claims that the specification of test cases is also doable by a DE. The goal of RQ 4 is to assess whether DEs indeed think that they are capable of specifying test cases, i.e., think that operating TDMS is easy.

Research questions 5 to 7 deal with the maintenance of declarative process models.
Research Question 5 (RQ 5): In our theoretical work (Zugal, Pinggera, & Weber, 2012) we postulate that the adoption of test cases has a positive effect on the cognitive load of process modelers. The goal of RQ 5 is to investigate whether the adoption of test cases significantly lowers the cognitive load on the process modeler conducting the change.
Research Question 6 (RQ 6): It is known from experiments conducted in the domain of software engineering that having test cases at hand improves perceived quality (Marchenko, Pekka, & Ihme, 2009). However, it is not clear yet whether similar effects are in place for test cases for declarative process models. Hence, the goal of RQ 6 is to assess whether the adoption of test cases significantly improves the perceived quality of the adapted process model.
Research Question 7 (RQ 7): As introduced in the background section, test cases provide an automated way of validating the process model. Thus, a positive influence on the quality of process models can be expected. The goal of RQ 7 is to investigate whether the adoption of test cases significantly improves the quality of the adapted process model.

DEFINITION AND PLANNING OF THE CASE STUDY

In the following, we report from a case study in which we look into the impact of TDM on the communication between MB and DE in real-life modeling sessions, i.e., investigate RQ 1 to RQ 4.

Case Study Methodology

The modeling methodology, as proposed by TDM, can be seen as collaborative approach. It assumes that the process model is created in an iterative process that requires intense communication between MB and DE. To investigate the communication, we follow the CoPrA approach for the analysis of collaborative modeling sessions (Seeber, Weber, & Maier, 2012).

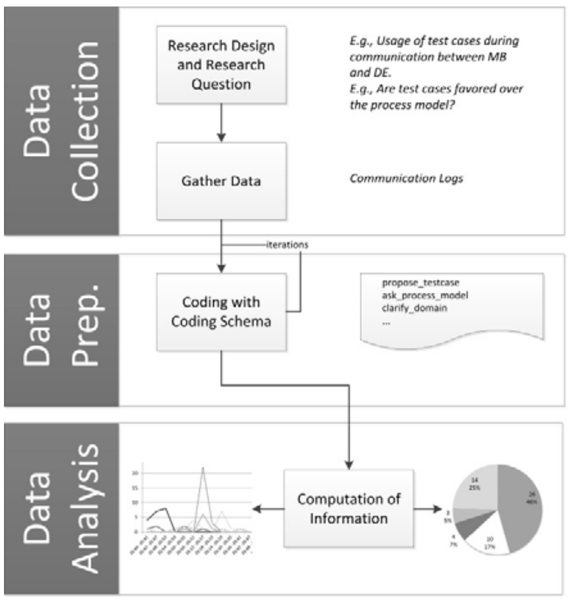

Data Collection
Research Design and Research Question
E.g., Usage of test cases during communication between MB and DE.
E.g., Are test cases favored over the process model?
Gather Data
Communication Logs
iterations
Data Prep.
Coding with Coding Schema
propose_testcase
ask_process_model
clarify_domain
...
Data Analysis
Computation of Information

Figure 6. Analysis technique for collaboration processes, adapted from (Seeber, Weber, & Maier, 2012)

As illustrated in Figure 6, the CoPrA approach consists of three phases. First, in the data collection phase, the research question and design are fixed. Based on the design, data is collected, e.g., by recording communication protocols. In the context of this work, data refers to the communication protocols recorded in modeling sessions. Second, in the data preparation phase, data is coded according to a coding schema. In this work, communication protocols first had to be transcribed to make them amenable for coding. Third, in the last phase, characteristics such as the distribution of codes can be analyzed. In this work, for instance, we analyzed how often the DE and MB referred to test cases. For a detailed description of the CoPrA approach, we refer the interested reader to (Seeber, Weber, & Maier, 2012).

Case Study Design

Based on research questions RQ 1 to RQ 4, the design of the case study was elaborated. The overall design of the case study consists of three phases. In the first phase, demographic data, such as age, familiarity with computers and experience in process modeling are collected. TDM assumes that the DE is a specialist in his/her domain, but is not trained in process modeling. Hence, these assessments are required to ensure that the DEs participating in the case study comply with this profile. In the second phase, the modeling sessions take place. For half of the subjects, a MB trained in TDM leads the modeling session. For the other half of the subjects, the MB conducts the modeling session using a declarative modeling editor only. During the modeling session, three data capturing mechanisms are used. To capture communication, audio and video data is recorded. In addition, TDMS is employed to gather the created process models and test cases. To ensure that the results are not biased by unfamiliarity with the usage of TDMS, it was operated by the MB only. Finally, in the third phase, the *Perceived Ease of Use* scale of the *Technology Acceptance Model (TAM)* (Davis, 1986) is presented to the DE in order to investigate RQ 4 (TDMS is easy to use). To address the research questions RQ 1 to RQ 3, we developed a coding scheme to code the transcribed communication logs. In particular, we used a subset of the collaboration patterns by Rittgen (Rittgen, 2007) to describe the communication between domain, DE, MB, test cases and process model (cf. Figure 4). As summarized in Table 1, we differentiate between asking questions (*ask*), answering questions (*clarify*), proposing changes

ACTION	CODING	EXAMPLE
Person p states a question q without referring to a test case or the process model.	*ask_domain (p, q)*	*"So you need to connect the mast with the sailing ship?"*
Person p states a question q regarding the notation.	*ask_notation (p,q)*	*"Is this a precondition?"*
Person p states a question q and refers to the process model.	*ask_process_model (p,q)*	*"This one too"? (points at process model)*
Person p states a question q and refers to a test case.	*ask_test_case (p,q)*	*"Then we start here." (points at test case)*
Person p gives answer a to question q without referring to a test case or the process model.	*clarify_domain (p, q, a)*	*"Yes, at least once and at most ten times."*
Person p gives answer a to question q regarding the notation.	*clarify_notation (p, q, a)*	*"This arrow indicates a precondition."*
Person p gives answer a to question q and refers to the process model.	*clarify_process_model (p, q, a)*	*"Here, we are currently…" (points at process model)*
Person p gives answer a to question q and refers to a test case.	*clarify_test_case (p, q, a)*	*"This happens up here, just before…" (points at test case)*
Person p makes proposal pr without referring to a test case or the process model.	*propose_domain (p, pr)*	*"Then, I have to contact the agency."*
Person p makes proposal pr and refers to the process model.	*propose_process_model (p, pr)*	*"Here, you have to unwrap the mainsail." (points to process model).*
Person p makes proposal pr and refers to a test case.	*propose_test_case (p, pr)*	*"Then, here, we went on with…" (points to test case)*
Person p expresses consent to proposal pr.	*support (p, pr)*	*"Yes, right."*

Table 1. Coding schema

(*propose*) and expressing consent to a proposal (*support*). Orthogonally, we distinguish whether MB and DE refer to the process model when talking (*process_model*), refer to a test case when talking (*test_case*) or just talk freely without referring to the process model or a test case (*domain*). To assess whether DEs take into account formal properties of the process model (cf. RQ 2), we added codes *ask_notation* to code situations where the DE asked about the modeling notation. For answering questions about the modeling notation, we used *clarify_notation*.

For the operationalization of this setup, we relied on the capabilities of TDMS. As TDMS was implemented as an experimental workflow activity of Cheetah Experimental Platform (CEP) (Pinggera, Zugal, & Weber, 2010), it could be seamlessly integrated in the experimental workflow. In other words, CEP guided MB and DE through the modeling sessions. Data was collected automatically, ensuring that each modeling session, the collected demographic data and TAM survey was stored as a separate case of the case study.

PERFORMING THE CASE STUDY

In the following, we will describe how the case study was performed and investigate research questions RQ 1 to RQ 4. Besides the elaboration of the case study design, the preparatory phase included the configuration of CEP, acquisition of appropriate devices for capturing audio and video, training the MB in TDM as well as the screening for potential DEs. Before the case study was started, a small pretest was conducted to ensure that the collected data is amenable for the envisioned analysis. After minor adaptations (e.g., other software for video capturing), the case study was started. All in all, eight DEs participated in the study. Each of them was asked to describe a process from a domain they were familiar with. In four modeling sessions, the TDM methodology was adopted, i.e., in addition to the process model, test cases were developed, validated and discussed. In the other four modeling sessions, the MB used a declarative process modeling editor only, no test cases were provided[3]. To ensure that the TDM methodology was properly adopted and results were not influenced by lacking familiarity with TDMS or declarative process models, the MB underwent intensive training in declarative process models, applying TDM and using TDMS. During the modeling sessions, TDMS was operated by the MB only to avoid the DE's different levels of tool knowledge to influence results. Each of these modeling sessions lasted between 19 and 32 minutes, in total 3 hours and 32 minutes of modeling were captured.

[3] Used material as well as collected data can be downloaded from: www.zugal.info/case_study/tdm

	Min	**Max**	**Mean**
Familiarity with computers	3	5	4.375
Familiarity with domain	3	5	4.75
Familiarity with BPM	1	2	1.125
Familiarity with declarative BPM	1	2	1.125

Table 2. Demographic data

After the modeling sessions had been finished, we checked whether the participating DEs actually met the targeted profile, i.e., they had to be experts in their domain, but should not be familiar with business process management. We used a 5-point Likert scale to test for familiarity with computers in general, the domain, BPM as well as declarative BPM. The scale ranged from *"Disagree"* (1) over *"Neutral"* (3) to "*Agree*" (5).[4] As summarized in Table 2, the case study's subjects fit the targeted profile of a DE. Involved domains were quite diverse and included, e.g., tax audition, creation of class schedules, sailing or renovating of buildings. In addition, we assessed the age of the participants. Two DEs were between 18 and 29, one DE between 30 and 45, two DEs between 45 and 60 and three DEs were over 60. Finally, we also assessed the familiarity with computers. As summarized in Table 2, all participants indicated strong accordance, canceling out that subjects were unfamiliar with computers.

In the following, the recorded communication protocols of all sessions were transcribed, resulting in a text

[4] The study was conducted with native German speakers; hence scale names were translated to avoid misunderstandings. The original scale names were: *"stimmt nicht"*, *"stimmt wenig"*, *"stimmt mittelmäßig"*, *"stimmt ziemlich"* and *"stimmt sehr"*.

document with 26,299 words uttered in 1,267 statements. Subsequently, the codes from Table 1 were used to code the transcripts. As indicated in Table 1, not all codes could be identified by looking at the transcripts only. For instance, for properly identifying code *ask_process_model*, the coder has to know whether the person referred to the process model. For this reason, also the videos of the modeling sessions were taken into account while coding. To get an overview of the coding, a summary is listed in Table 3. In general, it can be said that the amount of codes used in total is similar, i.e., 381 for TDM, 423 for conventional modeling. However, the distribution of codes shows differences. Unsurprisingly, codes that refer to test cases were not used for the transcripts of conventional modeling sessions. Interestingly enough, less asking and clarifying occurred in TDM sessions (84 versus 129 *ask*, 101 versus 131 *clarify*). Please note that the amount of *ask* statements does not necessarily need to coincide with the amount of *clarify* statements. In fact, several *clarify* statements may follow an *ask* statement. For instance, in some cases a question was picked up later and clarified in more depth. A further interesting relation can be found between the number of times questions about the notation were posed (*ask_notation*) and the number of times statements referred to the process model. 66 statements referred to the process model, but only once a DE uttered a question about the notation. Knowing that the participating DEs did not have any experience in process modeling (cf. Table 2), this finding is surprising. In the following, we will discuss the research questions in the light of the collected data.

Code	**TDM**	**Conv.**
ask_domain	45	109
ask_notation	0	1
ask_process_model	5	9
ask_test_case	34	0
clarify_domain	63	110
clarify_notation	0	3
clarify_process_model	4	18
clarify_test_case	34	0
propose_domain	113	95
propose_process_model	0	30
propose_test_case	26	0
support	57	48
Total	381	423

Table 3. Total codes used

RQ 1: Test Cases Provide an Additional Communication Channel

One of the basic claims of TDM is that test cases provide an additional communication channel between DE and MB. In other words, test cases serve as discussion basis. To investigate this claim, we counted how often MB and DE referred to a test case during the modeling session. Only those statements that unmistakably referred to a test case were counted. For the identification of such statements we used two criteria. Either, the test case was explicitly mentioned, e.g., *"So, now we are talking about the positive case"*. Or, when the transcript itself did not reveal whether the discussion was revolving around a test case, we consulted the video. Thereby, we checked whether the person pointed at a test case. If so, the statement was considered to be referring a test case.

Considering *ask*, in total 84 statements were uttered during TDM modeling sessions. 34 (40%) of them were referring to a test case. Regarding *clarify*, in total 101 statements were found. 34 (33%) of them were referring to a test case. Regarding *propose*, a total of 139 statements were found. 26 (19%) thereby referred to a test case. All in all, it can be said that for *ask* and *clarify*, a fair share of the communication was using test cases. Concerning *propose*, the proportion was clearly lower. In the following we would like to provide an explanation for this phenomenon. During the coding process, we found that DEs preferred to talk freely about their domain. In other words, their statements were not well-structured and included many aspects that could not be captured in the process model, such as time. In fact, this behavior is not surprising, as none of the participants was familiar with BPM (cf. Table 2). Speaking in terms of Figure 4, the DE requires the MB to translate knowledge in a form that can be modeled as test case or process model. Hence, it appears as if test cases cannot replace the abstraction skills of the MB. Still, 40% of the *ask* statements and 33% *clarify* statements used test cases. Consequently, we argue that test cases are apparently able to provide an additional communication channel. In other words, we can positively answer RQ 1: test cases do provide an additional communication channel. Please note that the observation that test cases cannot replace the abstraction skills of the MB does not contradict TDM. Rather, test cases should be modeled by DE and MB together. Test cases aim to improve the communication, but are not intended to replace the MB.

RQ 2: DEs Favor Test Cases over the Process Model for Communication

Even if test cases provide an additional communication channel, it is not clear yet whether it will be accepted for communication. TDM claims that test cases are easier to understand for DEs, hence DEs will favor test cases over process models for communication. To investigate this claim, we look at the collected data from two perspectives. First, we look at the ratio of test case based communication versus process model based communication *within* TDM modeling sessions. Then, we will compare the communication behavior of TDM and conventional modeling.

Regarding *ask*, in TDM modeling sessions 84 statements were uttered. 45 (54%) were classified as general statements, 5 (6%) referred to the process model, while 34 (40%) referred to a test case. A similar situation can be found for *clarify*. Here, 101 statements were transcribed, 63 (62%) were classified as general, 4 (4%) referred to the process model, while 34 (34%) referred to a test case. Finally, for *propose*, from a total of 139 statements, 113 (81%) were classified as general, 0 (0%) referred to the process model, whereas 26 (19%) referred to a test case. Interestingly, for any of these code categories a similar pattern can be found. Most communication happens in a general form (54 % to 81%), without referring to the process model or a test case. In addition, a noticeable share of the communication involves test cases (19% to 40%), while almost no communication is conducted with respect to the process model (0% to 6%). Especially interesting here is the fact that TDM provides test cases *as well as* the process model. Even though the idea is to focus on test cases for communication, such behavior cannot be enforced. In fact, TDMS provides test cases and process model side-by-side. In this light, the presented numbers make even a stronger case for test cases.

In the following, we will complement these findings by comparing TDM with conventional modeling. With respect to *ask*, 5 (6%) statements referred to the process model in TDM, 9 (8%) in conventional modeling. Regarding *clarify*, 4 (4%) referred to the process model in TDM, 18 (14%) in conventional modeling. Finally, for *propose*, 0 (0%) statements in TDM referred to the process model, in conventional modeling 30 (24%) could be found. Apparently, supplying test cases seems to distract attention from the process model, indicating that test cases are favored over process models. Hence, RQ 2 can be answered positively: the collected data suggests that test cases are favored over the process model in communication.

RQ 3: Test Cases Foster Communication

To answer RQ 3, we start by defining what fostering communication means and how it can be measured. The basis for our assessment is provided by Hoppenbrouwers et al. (Hoppenbrouwers, Lindemann, & Proper, 2006), who describe modeling as a *"dialogue of questioning and answering"*. Indeed, we could exactly observe this behavior while coding the communication protocols. Initially, the DE proposes a statement. If the statement is clear enough, the MB is able to directly reflect it in a test case or the process model. Otherwise, the MB has to ask the DE for clarification until the information is clear enough. Hence, we assume that the less *ask* statements and, in turn, *clarify* statements are required, the more efficient the communication.

To assess whether the adoption of TDM indeed foster communication, we counted the number of *ask*, *clarify* and *propose* statements. In TDM sessions, in total 84 *ask* statements were uttered, in conventional modeling sessions 118. Similarly, in TDM sessions 101 *clarify* statements could be observed, in conventional modeling sessions 128. These results suggest that less *ask* and *clarify* statements were observed in TDM modeling sessions. Still, the reason for the difference could be lead back to a different amount of total statements. In other words, the absolute amount of statements may differ, but not the relative difference. To cancel out this influence, we also counted the number of *propose* statements and computed the relative occurrence of *ask* and *clarify* with respect to *propose* statements. In TDM modeling sessions in total 139 *propose* statements were found, for conventional modeling sessions 125. Hence, in TDM modeling sessions, 0.60 *ask* statements were uttered per propose statement. In conventional modeling, this value increased to 0.94 times *ask* per *propose*. Similarly, 0.72 *clarify* statements were found per *propose* in TDM sessions, 0.97 for conventional modeling. All in all, the total amount of *ask* and *clarify* statements as well as relative amount of *ask* and *clarify* statements per *propose* are lower. Hence, we argue that also RQ 3 can be answered positively: the adoption of TDM appears to have a positive influence on communication by making communication more precise.

RQ 4: Operating TDMS is Easy

The last research question is partly related to TDM and partly related to TDMS. TDM claims that test cases are easier to understand than a process model. Similarly, TDMS claims the graphical user interface for test cases it is easy to use. To assess whether DEs would agree to these statements, we asked the participating DEs to fill out the *Perceived Ease of Use* scale from the Technology Acceptance Model (TAM) (Davis, 1986) after the modeling session. The *Perceived Ease of Use* scale consists of six 7-point Likert items, ranging from *"Extremely likely"* (1) over *"Neither Likely nor Unlikely"* (4) to *"Extremely Unlikely"* (7). On average, the DEs responded with 1.9, which approximately relates to *"Quite Likely"* (2). Hence, we conclude that the participating DEs find it "quite likely" that it would be easy to learn and operate TDMS.

EXPERIMENTAL DEFINITION AND PLANNING

So far we examined how TDM affects communication. In the following, we report on a controlled experiment that investigates the impact of test cases on the maintainability of declarative process models (RQ 5 to RQ 7). Descriptions are deliberately kept short; detailed information about the setup can be found in (Zugal, Pinggera, & Weber, 2011c).

Hypotheses. The hypotheses to be tested are directly derived from the research questions.
Hypothesis H1 (RQ 5): The adoption of test cases significantly lowers the cognitive load on the process modeler conducting the change.
Hypothesis H2 (RQ 6): The adoption of test cases significantly improves the perceived quality of the adapted process model.
Hypothesis H3 (RQ 7): The adoption of test cases significantly improves the quality of changes conducted during maintenance.

Subjects. The targeted subjects should be at least moderately familiar with business process management and declarative process modeling notations. We are not targeting modelers who are not familiar with declarative process models at all, since we expect that their unfamiliarity blurs the effect of adopting test cases as they have to struggle too much with the notation itself.

Objects. The objects of our study are two change assignments, each one performed on a different declarative process model[5]. Both models consisted of 7 activities, model 1 contained 7 constraints and model 2 contained 8 constraints. Please note that declarative models of such size are far from trivial, as the interplay of several constraints quickly becomes complex. To cancel out the influence of domain knowledge (Khatri, Vessey, Ramesh, & Park, 2006), we labeled the models' activities by letters (e.g., A to H).

[5] The material used for this study can be downloaded from: http://www.zugal.info/experiment/tdm

Factor and Factor Levels. Our experiment's factor is the adoption of test cases, i.e., whether test cases are provided while conducting the changes to the process model or not. Thus, we define the factor to be adoption of test cases with factor levels test cases and absence of test cases.

Response Variables. In order to test the hypotheses formulated above, we define the following response variables: 1) cognitive load, 2) perceived quality and 3) quality of the process model. To measure cognitive load, we employed a 7-point Likert scale, ranging from *"Very low"* (1) over *"Medium"* (4) to *"Very high"* (7). As detailed in (Paas, Tuovinen, Tabbers, & Van Gerven, 2003), employing Likert scales for measuring cognitive load has been shown to be reliable and is widely adopted. To assess perceived quality, we asked subjects to self-rate the quality, i.e., correctness, of the resulting model. To assess the quality of the process model, we assessed whether subjects properly conducted the changes without violating invariants that were defined for the initial model. For a detailed definition of the quality metric, we refer to (Zugal, Pinggera, & Weber, 2011c).

Instrumentation and Data Collection Procedure. We rely on TDMS for non-intrusive data collection. TDMS, as detailed in (Zugal, Pinggera, & Weber, 2011b), enables to investigate the maintenance tasks in detail by replaying the logged commands step-by-step.

PERFORMING THE EXPERIMENT

This section deals with the experiment's execution. First, operational aspects, i.e., how the experiment has been executed, are covered. Then, data is analyzed.

	N	Minimum	Maximum	Mean
Cognitive load with test cases	12	2	7	4.33
Cognitive load without test cases	12	4	7	5.75
Cognitive load overall	24	2	7	5.05
Perceived quality with test cases	12	4	7	6.00
Perceived quality without test cases	12	3	6	4.25
Perceived quality overall	24	3	7	5.12
Quality with test cases	12	20	23	22.33
Quality without test cases	12	19	23	21.92
Quality overall	24	19	23	22.13

Table 4. Descriptive statistics

Experimental Operation

Experimental Execution. The experiment was conducted in December 2010 at the University of Innsbruck in the course of a weekly lecture on business processes and workflows; all in all 12 students participated. To prepare the students, a lecture on declarative process models was held two weeks before the experiment. In addition, students had to work on several modeling assignments using declarative processes before the experiment took place. One week before the experiment, the concept of test cases and their usage was demonstrated. Immediately before the experiment, a short lecture revisiting the most important concepts of TDM and the experiment setup was held. The rest of the experiment was guided by CEP's experimental workflow engine (Pinggera, Zugal, & Weber, 2010), leading students through an initial questionnaire, two modeling tasks (one with the support of test cases and one without the support of test cases), a concluding questionnaire and a feedback questionnaire.

Data Validation. We screened the subjects for familiarity with DecSerFlow (Pesic, 2008) (the declarative process modeling language we use in our models), since our research setup requires subjects to be at least moderately familiar with DecSerFlow. We used a Likert scale with values ranging from *"Strongly disagree"* (1) to *"Neutral"* (4) to *"Strongly agree"* (7). The computed mean is 3.17 (slightly below average). For confidence in understanding DecSerFlow models a mean value of 3.92 was reached (approximately average). Finally, for perceived competence in creating DecSerFlow models, a mean value of 3.83 (approximately average) could be computed. Since all values range about average, we conclude that the participating subjects fit the targeted profile.

Data Analysis

In the following we describe the analysis and interpretation of data.

Descriptive Analysis. To give an overview of the experiment's data, Table 4 shows minimum, maximum and mean values of cognitive load, perceived quality and quality. The values shown in Table 4 suggest that the adoption of test cases lowers cognitive load, increases perceived quality and increases quality, thus supporting hypotheses *H1*, *H2* and *H3*. However, these observations are merely based on descriptive statistics. For a more rigid investigation, the hypotheses will be tested for statistical significance in the following.

Hypotheses Testing. Our sample was relatively small, thus we followed guidelines for analyzing small samples (Pett, 1997) and employed non-parametric tests. In particular, we used SPSS (Version 17.0) to carry out Wilcoxon Signed-Rank Test (Pett, 1997)[6].

Hypothesis H1: Applying Wilcoxon Signed-Rank Test for the response variable cognitive load yields a p-value of 0.010 (< 0.05), thus supporting *H1*.
Hypothesis H2: Applying Wilcoxon Signed-Rank Test for the response variable perceived quality yields a p-value of 0.005 (< 0.05), thus supporting *H2*.
Hypothesis H3: Applying Wilcoxon Signed-Rank Test for the response variable quality yields a p-value of 0.391 (> 0.05), thus rejecting *H3*.

Summing up, research questions concerning cognitive load (RQ 5) and perceived quality (RQ 6) could be answered positively. Regarding quality (RQ 7), data did not show a significant difference. Reasons, implications and conclusions are discussed in the following.

DISCUSSION

Up to now research questions RQ 1 to RQ 7 have been answered in isolation. Subsequently, we will integrate the findings to discuss the insights gained in this empirical investigation.

The findings of the case study seem quite positive so far. In fact, RQ 1 to RQ 4 could be answered all in favor of TDM and TDMS, respectively. It seems as if test cases are accepted as communication channel (RQ 1) and are favored over the process model (RQ 2). Furthermore, test cases decreased the amount of *ask* and *clarify* statements per *propose* (RQ 3). In addition, DEs indicated that they think learning to operate TDMS seems to be easy (RQ 4). In the following, we will underpin these findings with insights gained during data analysis. One of the central insights of the modeling sessions was about the way how DEs structured their information. In general, DEs seemed to prefer talking about their domain in the form of *sequential* actions, seen from their perspective. Indeed, 837 times words indicating sequential information, such as *"then"*, *"now"* or *"afterwards"* were used. Interestingly enough, this behavior could be observed across *all* modeling sessions. As this behavior occurred for TDM and conventional modeling sessions, we assume that it is the intuitive way for a DE to talk about a domain. If this was indeed the case, it would explain why test cases were well accepted. As argued in (Zugal, Pinggera, & Weber, 2012), test cases provide a *sequential*, instance-based view on a declarative process model. Contrariwise, a declarative process model rather provides *circumstantial* information. Consequently, it seems natural that DEs prefer test cases for communication.

TDM assumes that DEs are normally not able to read *formal* process models, as suggested in (Hoppenbrouwers, Lindemann, & Proper, 2006). To investigate whether this claim can be supported, we looked into all statements that referred to the process model. 66 (8%) times, either the DE or the MB referred to the process model. The analysis revealed all but one DE referred to activities, but ignored the constraints. Hence, it appears that DEs are normally not able to grasp the *formal* parts of a process model, but may access superficial information, such as activity names. Backup for this theory is also provided by the fact that none of the DEs had experience in declarative process modeling (cf. Table 2)

[6] As due to repeated measurements the variables are *not independent*, Wilcoxon Signed-Rank Test was chosen.

and only once a DE uttered a question about the notation (cf. Table 3). In other words, it seems implausible that DEs were able to extract more information from the process model than activity names.

Regarding the maintainability of declarative process models, we can conclude that the adoption of test cases has a positive influence on the cognitive load (RQ 5) and perceived quality (RQ 6). Especially interesting is that, even though quality could not be improved significantly (RQ 7), modelers have been more confident that they conducted the changes properly. This effect is also known from software engineering, where test cases improve perceived quality (Marchenko, Pekka, & Ihme, 2009; Beck, 1999). Indeed also the follow-up discussion with the students after the experiment revealed that students with software development background experienced this similarity, further substantiating our hypotheses on a qualitative basis.
Regarding quality (RQ 7), no statistically significant differences could be observed. This raises the question whether there is no impact of test cases on quality at all or if the missing impact can be explained otherwise. To this end, a detailed look at the distribution of quality offers a plausible explanation. The overall quality is very high, the quality measured on average is 22.13 out of a maximum of 23 (cf. Table 4). Thus, approximately 96% of the questions have been answered correctly / tasks have been carried out properly. Put differently, the overall quality leaves almost no room for improvements when adopting test cases—in fact, the sample's standard deviation is very low (1.39). Since data "points towards" the positive influence of test cases on quality (i.e., the mean value is higher, cf. Table 4) and due to the low variance it seems reasonable to assume that a positive correlation exists, however, the overall high quality blurs expected effects. To test this assumption, a replication with more complex and thus more challenging change tasks is planned. The increased complexity should result in a lower overall quality, thereby leaving room for improvements and thus allow distinguishing the effect of adopting test cases. In other words, it can be assumed that a certain level of complexity is required in order for test cases to show positive effects.

LIMITATIONS

Even though the investigation seems quite promising, the presented results are to be viewed in the light of limitations. First and foremost, the sample size is a threat to the generalization of results. All in all, only 20 subjects could be acquired for the investigation, i.e., 8 subjects for the case study and 12 subjects for the controlled experiment. Furthermore, RQ 1 to RQ 4 rely on a qualitative data basis—the presented numbers are of descriptive nature and are not strong enough for inferential statistics. In addition, the modeling sessions were rather short, on average a modeling session lasted 26 minutes and 30 seconds. Another limitation is the focus of TDM on control flow. Finally, for RQ 5 to RQ 7, students served as subjects. Even though it has been shown that for software engineering, students may provide an adequate model for the professional population (Höst, Regnell, & Wohlin, 2000), there exist also studies that show significant differences between students and professionals (e.g., (Arisholm & Sjøberg, 2004)). Hence, the presented results can only be generalized to limited extent.

RELATED WORK

TDM adopts well-established practices from software engineering and adapts them to the application in declarative process models. Hence, related work can be found in the

fields of software engineering as well as business process management. Most notably is the work of Ly et al. (Ly, Rinderle, & Dadam, 2008), which also focuses on the validation of the process model, however, in contrast to our work, adaptive process management systems are targeted instead of declarative ones. With respect to process validity, work in the area of process compliance checking should be mentioned, e.g., (Awad, Decker, & Weske, 2008). In contrast to our work, understandability of declarative languages is not of concern; the focus is put on imperative languages.
Another related stream of research is the verification of declarative process models. With proper formalization, declarative process models can be verified using established formal methods (Pesic, 2008). Depending on the concrete approach, a-priori (e.g., absence of deadlocks) (Pesic, 2008) or a-posteriori (e.g., conformance of the execution trace) (van der Aalst, de Beer, & van Dongen, 2005) checks can be performed. While these approaches definitively help to improve the syntactical correctness and provide semantical checks a-posteriori, they do not address understandability and maintainability issues.
Related are also so-called scenario-based approaches to process modeling, where scenarios specify a certain aspect of a business process similar to a test-case (e.g., (Glinz, Seybold, & Meier, 2007; Fahland, 2010)). However, existing approaches focus on imperative process modeling notations, e.g., Petri Nets, whereas our approach is clearly focused on declarative process modeling notations.
With respect to empirical evaluation, experiments investigating the effect of Test Driven Development (TDD), i.e., interweaving software development and testing, like the TDM methodology interweaves modeling and testing, are of interest. (Marchenko, Pekka, & Ihme, 2009) conducted a long-term case study which showed that the adoption of TDD increases perceived quality. In addition, developers stated that changes could be conducted easier when TDD was adopted. With respect to code quality, the situation appears not to be entirely clear. For instance, (George & Williams, 2004; Edwards, 2003) report from controlled experiments that showed increased code quality through the adoption of TDD. Contrariwise, experiments conducted in (Erdogmus, Maurizio, & Torchiano, 2005; Pančur, Ciglarič, Trampuš, & Vidmar, 2003) could not show any significant difference between TDD and test-after coding. More generally, benefits of TDD with respect to quality and perceived quality could so far mostly be shown for industrial settings—in semi-industrial or academic context, the situation seems less clear (Siniaalto, 2006). In this sense, results reported in this work seem plausible, as except for RO 7, a positive influence of TDM could be shown.

CONCLUSION

In this work, TDM was investigated from two different angles. In particular, the impact of TDM on communication was explored in a case study. In addition, the impact of TDM on model maintainability was examined in a controlled experiment. The conclusions to be drawn are as follows. First, test cases, as proposed in TDM, are accepted by DEs in modeling sessions. Collected data indicates that DEs favor test cases over the process model, given that both are offered at the same time. Apparently, the sequential nature of test cases provides an intuitive way for describing processes. In this vein, test cases also seem to make communication more efficient by reducing the number of *ask-clarify* cycles. Once scenarios have been captured in the form of test cases, they contribute to reducing cognitive load and increasing perceived quality during model maintenance. Regarding the impact on model quality, collected data is not conclusive. All in all, the merits of TDM that have been based so far mainly on theoretical considerations could be underpinned with empirical data. Still, it should not be forgotten to mention that the generalization of results is limited due to the small sample size. To weed out this weak spot, we are currently conducting replications for the controlled experiment as well as the case study.

ACKNOWLEDGEMENTS

This research is supported by Austrian Science Fund (FWF): P23699-N23. Gefördert aus Mitteln des vom Land Tirol eingerichteten Wissenschaftsfonds.

REFERENCES

Agrawal, H., Horgan, J., Krauser, E., & London, S. (1993). Incremental Regression Testing. *In: Proceedings of International Conference on Software Maintenance '93*, (pp. 348-357).

Arisholm, E., & Sjøberg, D. I. (2004). Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software. *IEEE Transactions on Software Engineering, 30*(8), 521-534.

Awad, A., Decker, G., & Weske, M. (2008). Efficient Compliance Checking Using BPMN-Q and Temporal Logics. *In: Proceedings of Business Process Management '08*, (pp. 326-341).

Beck, K. (1999). *Extreme Programming Explained: Embracing Change.* Addison-Wesley.

Beck, K. (2002). *Test Driven Development: By Example.* Addison-Wesley.

Davis, F. (1986). *A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results.* PhD Thesis, Sloan School of Management, Massachusetts Institute of Technology.

Dumas, M., van der Aalst, W., & ter Hofstede, A. (2005). *Process Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley-Interscience.

Edwards, S. H. (2003). Using Test-Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance. *In: Conference on Education and Information Systems: Technologies and Applications '03*, (pp. 421-426).

Erdogmus, H., Maurizio, M., & Torchiano, M. (2005). On the Effectiveness of the Test-First Approach to Programming. *IEEE Transactions on Software Engineering, 31*(1), 226-237.

Fahland, D. (2010). *From Scenarios to Components.* PhD thesis, Humboldt-Universität zu Berlin.

George, B., & Williams, L. (2004). A structured experiment of test-driven development. *Information and Software Technology, 46*(5), pp. 337-342.

Glinz, M., Seybold, C., & Meier, S. (2007). Simulation-Driven Creation, Validation and Evolution of Behavioral Requirements Models. *In: Proceedings of Model-Based Development of Embedded Systems '07*, (pp. 103-112).

Green, T., & Petre, M. (1996). Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions Framework'. *Journal of*

Visual Languages and Computing, 7(2), pp. 131-174.

Hoppenbrouwers, S., Lindemann, L., & Proper, E. (2006). Capturing Modeling Processes - Towards the MoDial Modeling Laboratory. *In: On the Move to Meaningful Internet Systems*, (pp. 1241-1252).

Höst, M., Regnell, B., & Wohlin, C. (2000). Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering, 5*(3), 201-214.

Khatri, V., Vessey, I., Ramesh, P., & Park, S. (2006). Understanding Conceptual Schemas: Exploring the Role of Application and IS Domain Knowledge. *Information Systems Research, 17*(1), pp. 81-99.

Lenz, R., & Reichert, M. (2007). IT support for healthcare processes - premises, challenges, perspectives. *Data & Knowledge Engineering, 61*(1), pp. 39-58.

Ly, L., Rinderle, S., & Dadam, P. (2008). Integration and verification of semantic constraints in adaptive process management systems. *Data & Knowledge Engineering, 64*(1), pp. 3-23.

Marchenko, A., Pekka, A., & Ihme, T. (2009). Long-Term Effects of Test-Driven Development A Case Study. *In: Proceedings XP '09*, (pp. 13-22).

Miller, G. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review, 63*(2), pp. 81-97.

Paas, F., Tuovinen, J. E., Tabbers, H., & Van Gerven, P. W. (2003). Cognitive Load Measurement as a Means to Advance Cognitive Load Theory. *Educational Psychologist, 38*(1), 63-71.

Pančur, M., Ciglarič, M., Trampuš, M., & Vidmar, T. (2003). Towards Empirical Evaluation of Test-Driven Development in a University Environment. *In: International Conference on Computer as a Tool '03*, (pp. 83-86).

Pesic, M. (2008). *Constraint-Based Workflow Management: Shifting Control to Users.* PhD thesis, TU Eindhoven.

Pesic, M., Schonenberg, H., Sidorova, N., & van der Aalst, W. (2007). Constraint-Based Workflow Models: Change Made Easy. *In: Proceedings of International Conference on Cooperative Information Systems '07*, (pp. 77-94).

Pett, M. (1997). *Nonparametric Statistics for Health Care Research: Statistics for Small Samples and Unusual Distributions.* Sage Publications.

Pinggera, J., Zugal, S., & Weber, B. (2010). Investigating the Process of Process Modeling with Cheetah Experimental Platform. *In: Proceedings of Empirical Research in Process-Oriented Information Systems '10*, (pp. 13-18).

Reichert, M., & Dadam, P. (1998). ADEPTflex: Supporting Dynamic Changes of Workflow without Losing Control. *Journal of Intelligent Information Systems, 10*(2), pp. 93-129.

Rittgen, P. (2007). Negotiating Models. *In: Proceedings of Conference on Advanced Information Systems Engineering '07*, (pp. 561-573).

Seeber, I., Weber, B., & Maier, R. (2012). CoPrA: A Process Analysis Technique to Investigate Collaboration in Groups. *In: Hawaii*

International Conference on System Sciences '12.

Siniaalto, M. (2006). *Test driven development: empirical body of evidence.* Technical report, ITEA, Information Technology for European Advancement.

van Bommel, P., Hoppenbrouwers, S., Proper, E., & van der Weide, T. (2006). Exploring Modelling Strategies in a Meta-modelling Context. *In: Proceedings of On the Move to Meaningful Information Systems '06*, (pp. 1128-1137).

van der Aalst, W. (2005). Case handling: a new paradigm for business process support. *Data & Knowledge Engineering, 53*(2), pp. 129-162.

van der Aalst, W., de Beer, H., & van Dongen, B. (2005). Process Mining and Verification of Properties: An Approach Based on Temporal Logic. *In: Proceedings of On the Move to Meaningful Internet Systems '05*, (pp. 130-147).

Weber, B., Reichert, M., & Rinderle, S. (2008). Change Patterns and Change Supports Features - Enhancing Flexibility in Process-Aware Information Systems. *Data & Knowledge Engineering, 66*(3), pp. 438-466.

Weber, B., Reijers, H., Zugal, S., & Wild, W. (2009). The Declarative Approach to Business Process Execution: An Empirical Test. *In: Proceedings of Conference on Advanced Information Systems Engineering '09*, (pp. 270-285).

Zugal, S., Pinggera, J., & Weber, B. (2011a). Assessing Process Models with Cognitive Psychology. *In: Proceedings of Enterprise Modelling and Information Systems Architectures '11*, (pp. 177-182).

Zugal, S., Pinggera, J., & Weber, B. (2011b). Creating Declarative Process Models Using Test Driven Modeling Suite. *In: Proceedings of Conference on Advanced Information Systems Engineering Forum '11*, (pp. 16-32).

Zugal, S., Pinggera, J., & Weber, B. (2011c). The Impact of Testcases on the Maintainability of Declarative Process Models. *In: Business Process Modeling, Development, and Support '11*, (pp. 163-177).

Zugal, S., Pinggera, J., & Weber, B. (2012). Toward Enhanced Life-Cycle Support for Declarative Processes. *Journal of Software: Evolution and Process, 24*(3), pp. 285-302.

Zugal, S., Pinggera, J., Weber, B., Mendling, J., & Reijers, H. (2011). Assessing the Impact of Hierarchy on Model Understandability—A Cognitive Perspective. *In: Proceedings of Experiences and Empirical Studies in Software Modelling '11*, (pp. 123-133).