Software Stories Guide

Albert Zündorf, Tobias George, Ruben Jubeh, Ulrich Norbisrath, Sabine Sachweh

Kassel University, Kassel University, Österreich, FH Dortmund

1 Motivation

Usual requirements engineering approaches [Pohl2010] try to cover the whole business process at once and come up with process descriptions containing a lot of rules, conditional actions, loops, etc, cf. [BPMN]. We consider such complex process descriptions a major problem as they are hard to understand for non IT persons, i.e. for the domain experts and the people that actively execute the modeled process. The situation is even worse concerning the data involved in the process. Standard requirements engineering comes up with glossaries, entity relationship models, and class diagrams. The later two are recognizable for IT people ownly. Usual people do not get a lot of information from a UML class diagram and thus they can not contribute to them nor identify missing parts or modeling faults.

Software Stories overcome these problems by focusing on concrete example scenarios. A Software Story addresses a single scenario not all possible scenarios, at once. Using examples from the every day live of the domain experts help them to express their knowledge about the process. If there are different cases, we use multiple different scenarios and Software Stories, one for each case. Whithin a Software Story, the domain experts may exemplify relevant data in the form it occurs to them. For example, they may add PDF forms, excel sheets, word documents, or GUI masks of tools they already use. For Software Stories it is perfect to give just concrete examples of e.g. one filled PDF form for a certain process step. Analyzing the concrete scnarios and the provided example data is done by IT experts in later software development steps. This is not the concern of the domain experts.

This paper examplifies the usage of Software Stories for requirements engineering and some user center design activities. We will first exemplify how we have used Software Stories for the development of a simple administrative tool at our research group. We will then discuss some important aspects and caveats of Software Stories and how Software Stories shall be used in a software development process.

2 Example for Current State Analysis

As an example we model the administrative process how bachelor and master theses are managed in our department or research group. The problem is somewhat "academic" as it does not stem from a typical business environment. However, the problem is typical for the kinds of problems addressed by software stories as multiple persons are involved at different points in time and the communication, coordination, and collaboration of these peoples currently lacks transparency and tool support. Thus we wanted to develop a workflow tool supporting us in the coordination of these processes.

First, we needed to do some requirements elicitation defining the theses workflow at our research group. Instead of BPMN notation we did the requirements elicitation with Software Stories. Figure 1 shows the first two steps of our first Software Story for the Theses Workflow Manager.

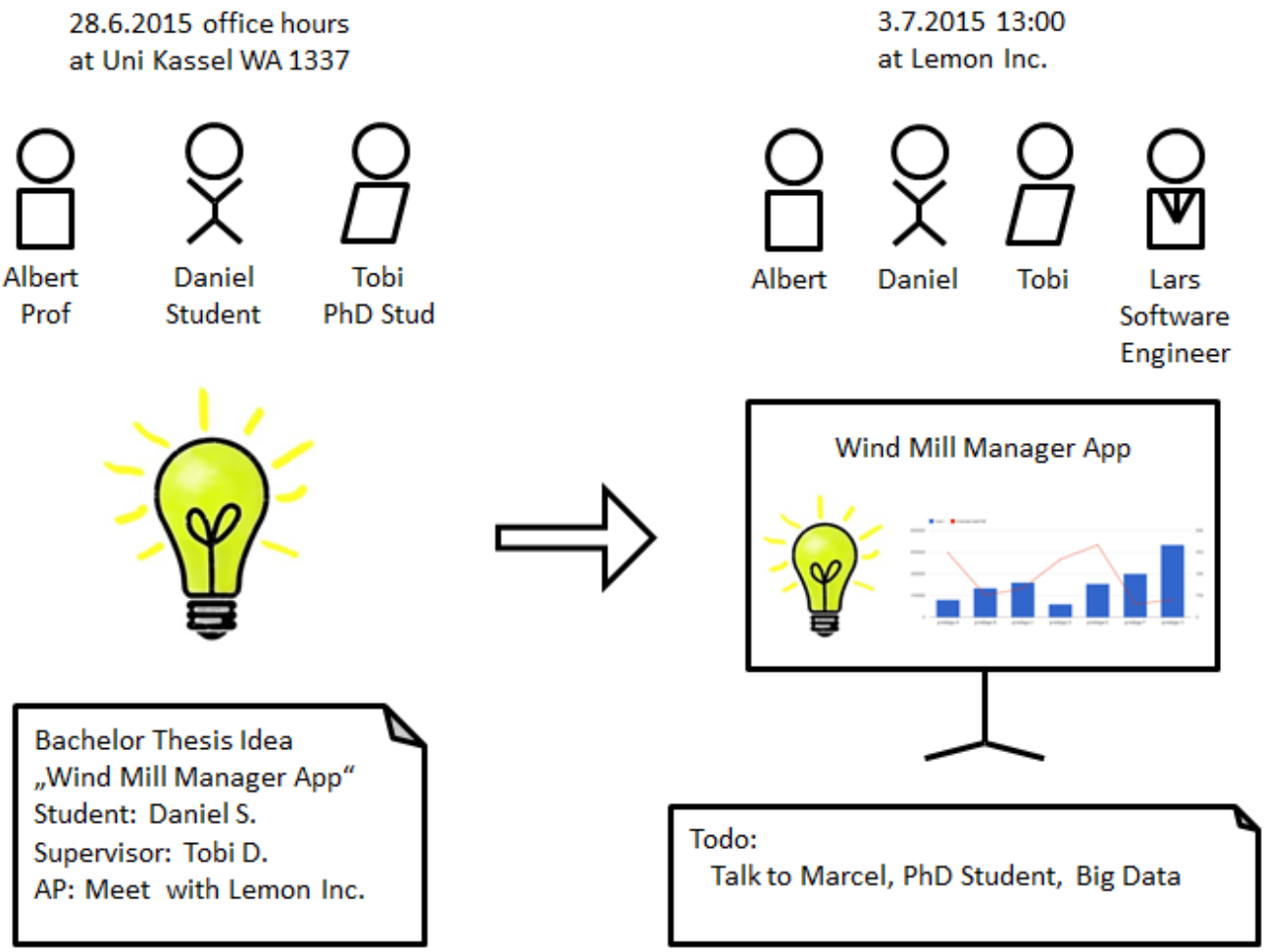


Figure 1: Software Story for Daniel's Bachelor Thesis, The start

A Software Story consists of a number of Steps. A Step describes a concrete situation and actions executed in that situation. A Step also describes when this step has been (or will be) executed and who participated in this step or who executed it. In Figure 1 the first step is shown on the left. It has been executed at June 28th 2015 at about 3pm. Daniel a computer science student at Kassel University met Professor Albert at Albert's office. Daniel wanted to do his Bachelor thesis under supervision of Albert. Daniel was student programmer at Lemon Inc. [1] in Kassel. Daniel was programming some Software managing Wind Mills for electrical power generation. Daniel wanted to do his Bachelor thesis in cooperation with Lemon Inc. in the context of his work at Lemon Inc. In our research group the actual supervision of bachelor thesis is usually done by PhD students. Thus, Albert asked Tobi, one of his PhD students to join the discussion. Daniel gave some more information on the context of his work at Lemon Inc. The topic seemed to be interesting and related to our research thus we agreed to supervise Daniel's Bachelor thesis. For Theses in cooperation with an enterprise, in our group a meeting with a supervisor provided by the enterprise is mandatory. Thus, Daniel got the task to organize such a meeting.

[1] Name changed by editors.

The second step of our example is shown on the right of Figure 1. On July 7th 2015 at 1pm Albert, Daniel, and Tobi met with Lars at Lemon Inc. Lars was a software engineer responsible for the Wind Mill Management [2] software developed at Lemon Inc. Daniel gave a short presentation on the content of the planned Bachelor thesis and Lars provided more details to the technical challenges. It turned out that Lemon Inc. was seeking for a Big Data solution to the recording and analysis of sensor data from huge numbers of wind mills. As Big Data is the special expertise of Marcel, another PhD student of our group, it was discussed to contact Marcel and ask him to supervise Daniel's thesis and to collaborate with Lemon Inc. on the Big Data issues.

[2] Thesis topic simplified and translated from German.

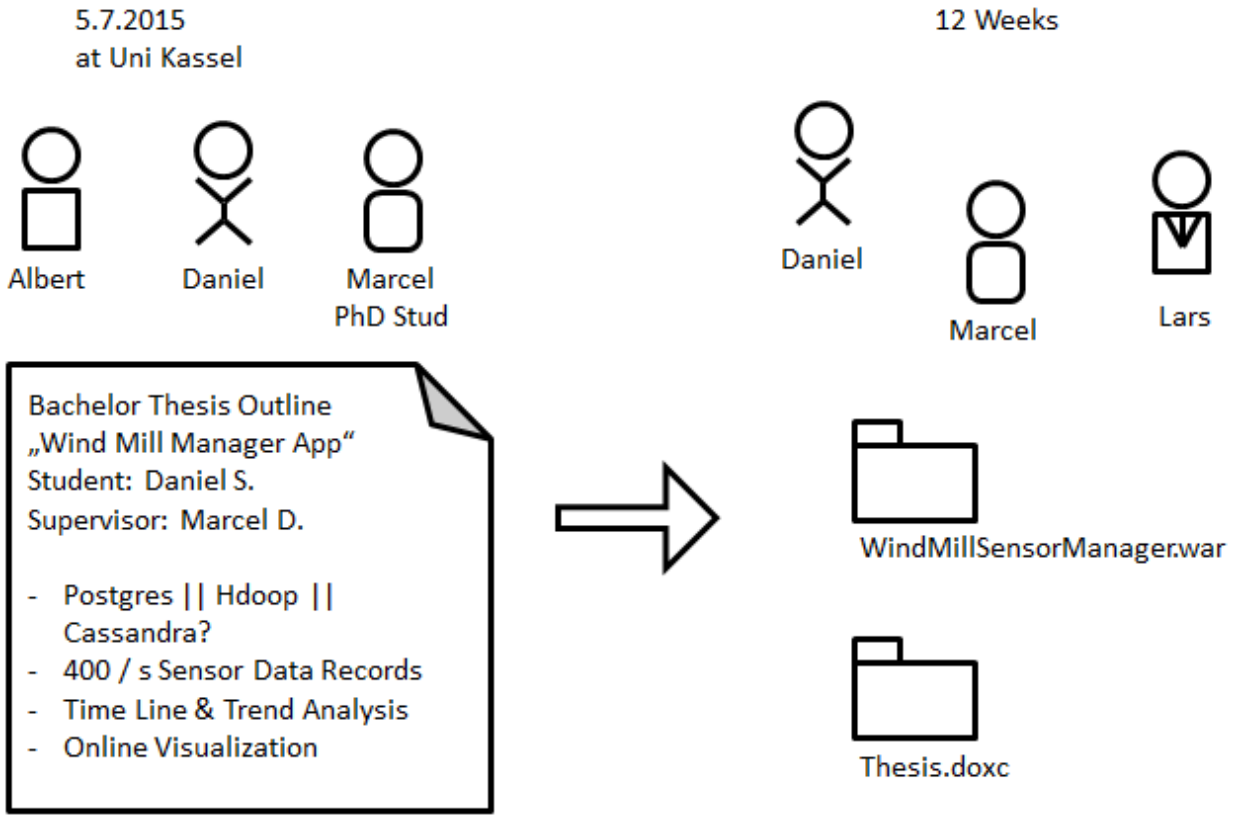


Figure 2: Software Story for Daniel's Bachelor Thesis, Doing It

Figure 2 shows on the left the meeting with Albert, Daniel, and Marcel on July 7th. Marcel was happy to help in this thesis and brought at lot of ideas to the table. In the thesis outline the most important topics were added as which database system would serve best for the thesis, how many data we expected, and what should be done with data. Another meeting with Daniel, Marcel, and Lars was scheduled in order to let Marcel and Lars known each other. Then the actual thesis work started.

The actual thesis work is shown on the right of Figure 2. This has been done mostly by Daniel. Marcel and Lars only assisted by giving directions for areas that need more investigation and how to organize work and how to structure the thesis itself. This lasted about 3 month. The outcome was the implementation of the Wind Mill Sensor Manager and the thesis document.

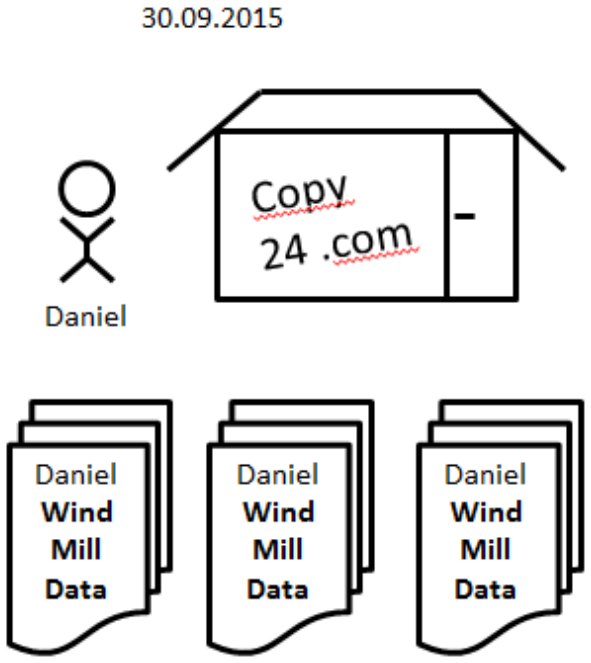


Figure 3: Software Story for Daniel's Bachelor Thesis, Printing

As shown in Figure 3, in the morning of September 30th Daniel went to the copy shop and 3 copies of his thesis were printed.

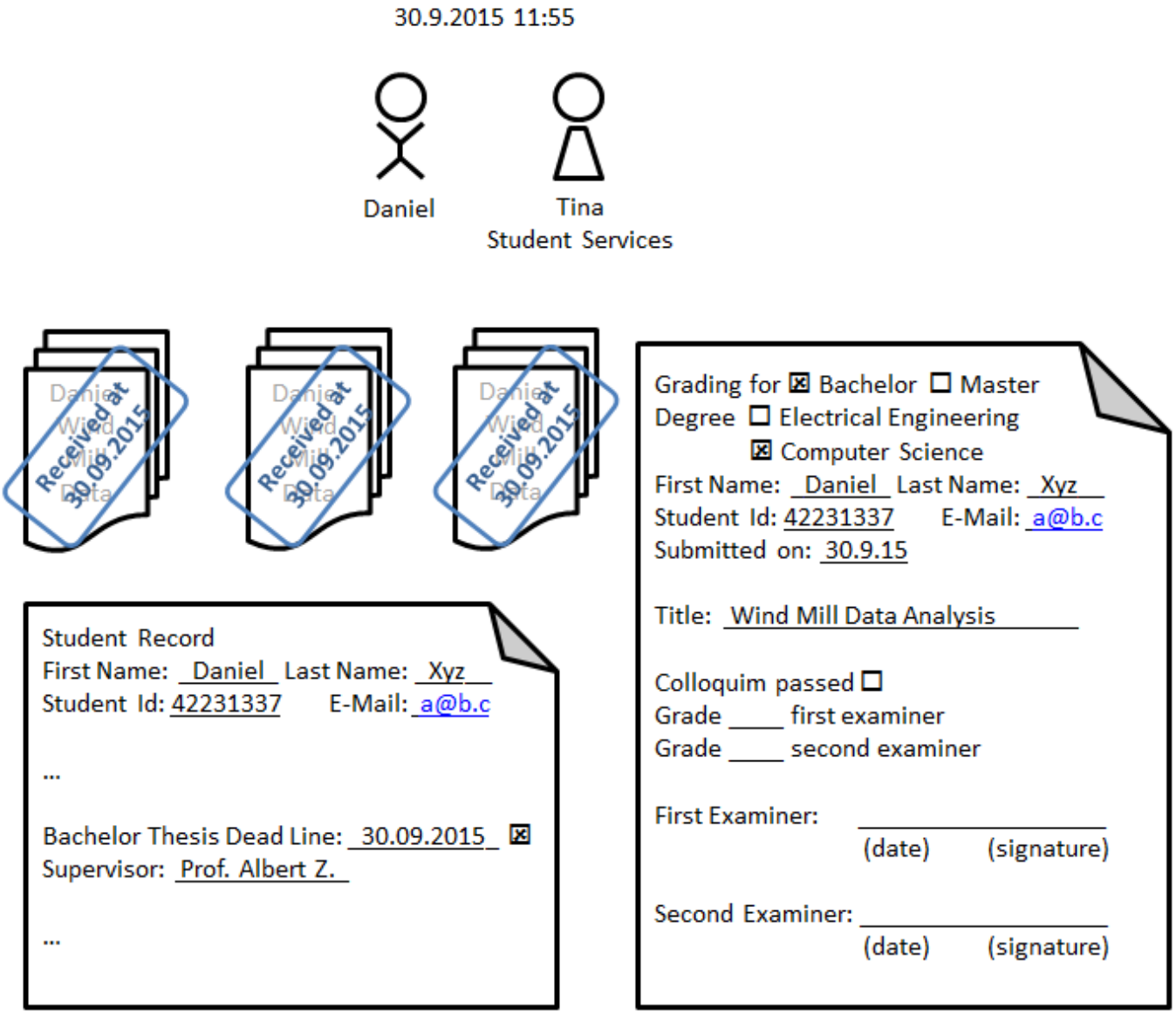


Figure 4: Software Story for Daniel's Bachelor Thesis, Submission

Then the difficult administrative process started. Just in time Daniel entered the Student Service Office of our department and handed in the three copies, cf. Figure 4. Tina is working at the Student Service. She stamped the three thesis copies with the date of their reception. Then, Tina fetched Daniel's Student Record and marked the dead line for the bachelor thesis as met. Finally Daniel and Tina filled in two copies of the Thesis Submission Form [3] of our department. Tina handed the stamped copies of thesis and the two grading forms to Daniel. Daniel had the responsibility to deliver the copies and the forms to his examiners.

[3] Forms simplified and translated from German.

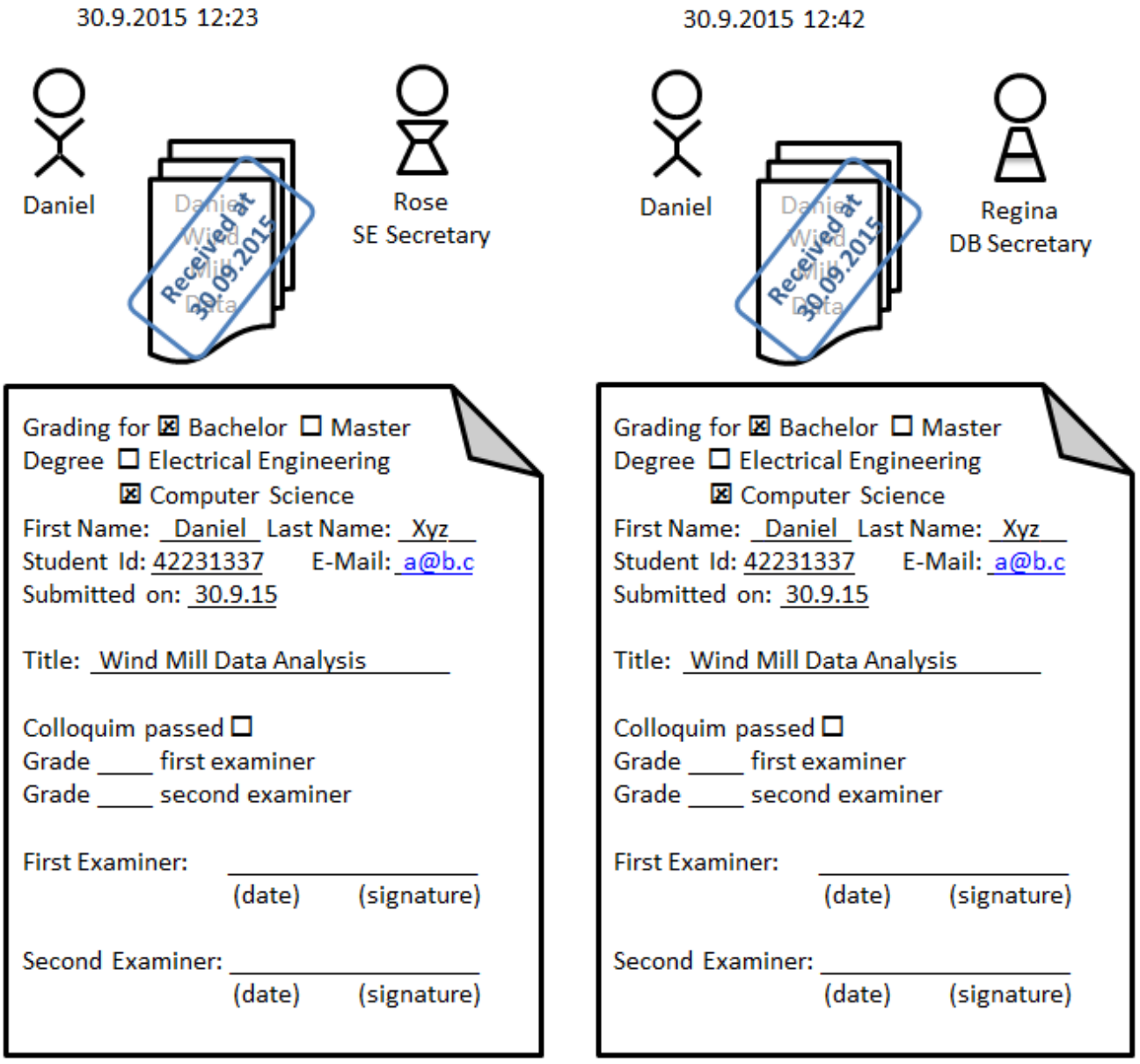


Figure 5: Software Story for Daniel's Bachelor Thesis, Examiners

Figure 5 shows how Daniel delivered one copy of his thesis and one copy of the grading form to Rose, the secretary of the Software Engineering group, and to Regina, the secretary of the database group, respectively. The professors of these groups, Albert and Lutz, are the two examiners for Daniel's thesis.

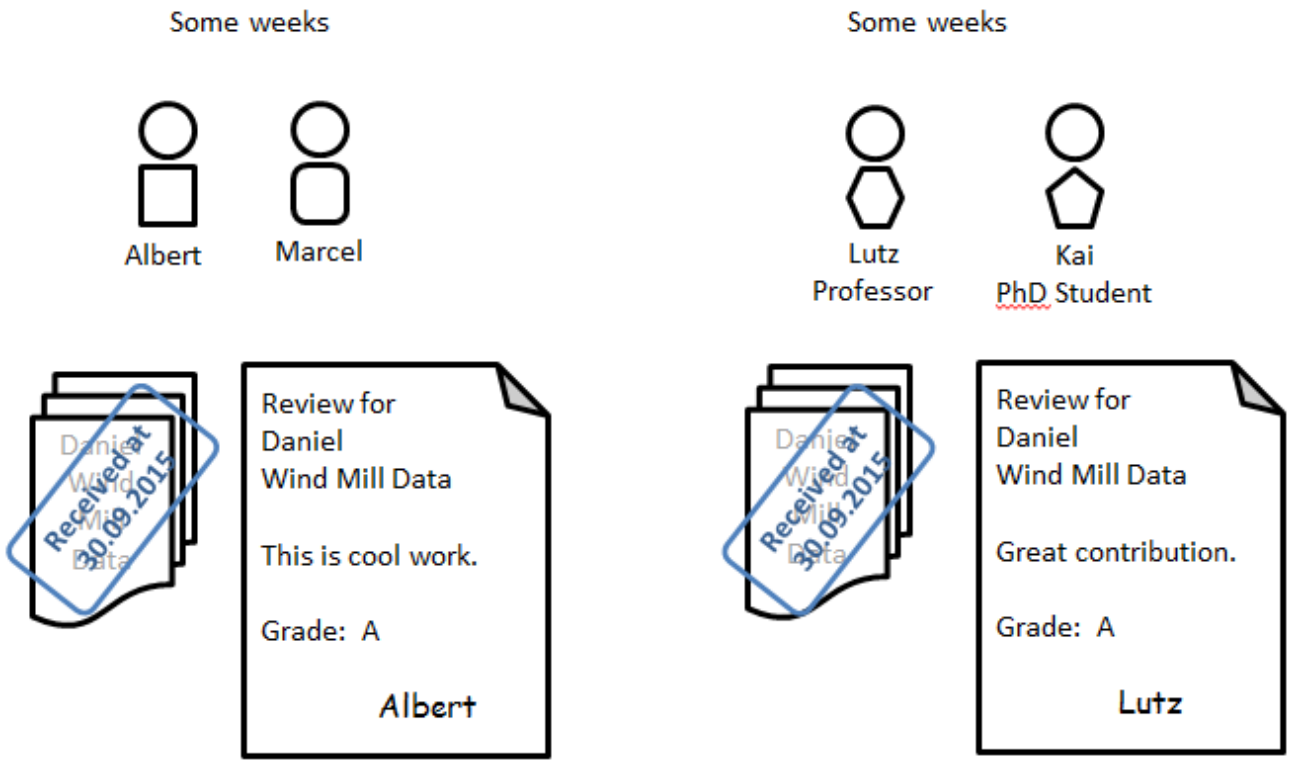


Figure 6: Software Story for Daniel's Bachelor Thesis, Examination

In Figure 6 Albert and Lutz read Daniel's thesis and wrote a review giving a grade for it. This has been done completely independent from each other. There are rumours that some professors do not read the theses them self but just ask their PhD students to do this. There are other rumours that sometimes the second examiner waits with his review until the first examiner sends his review as a guideline. Actually, in our department bachelor and master thesis reviews are not mandatory and sometimes the second reviewer does not produce one.

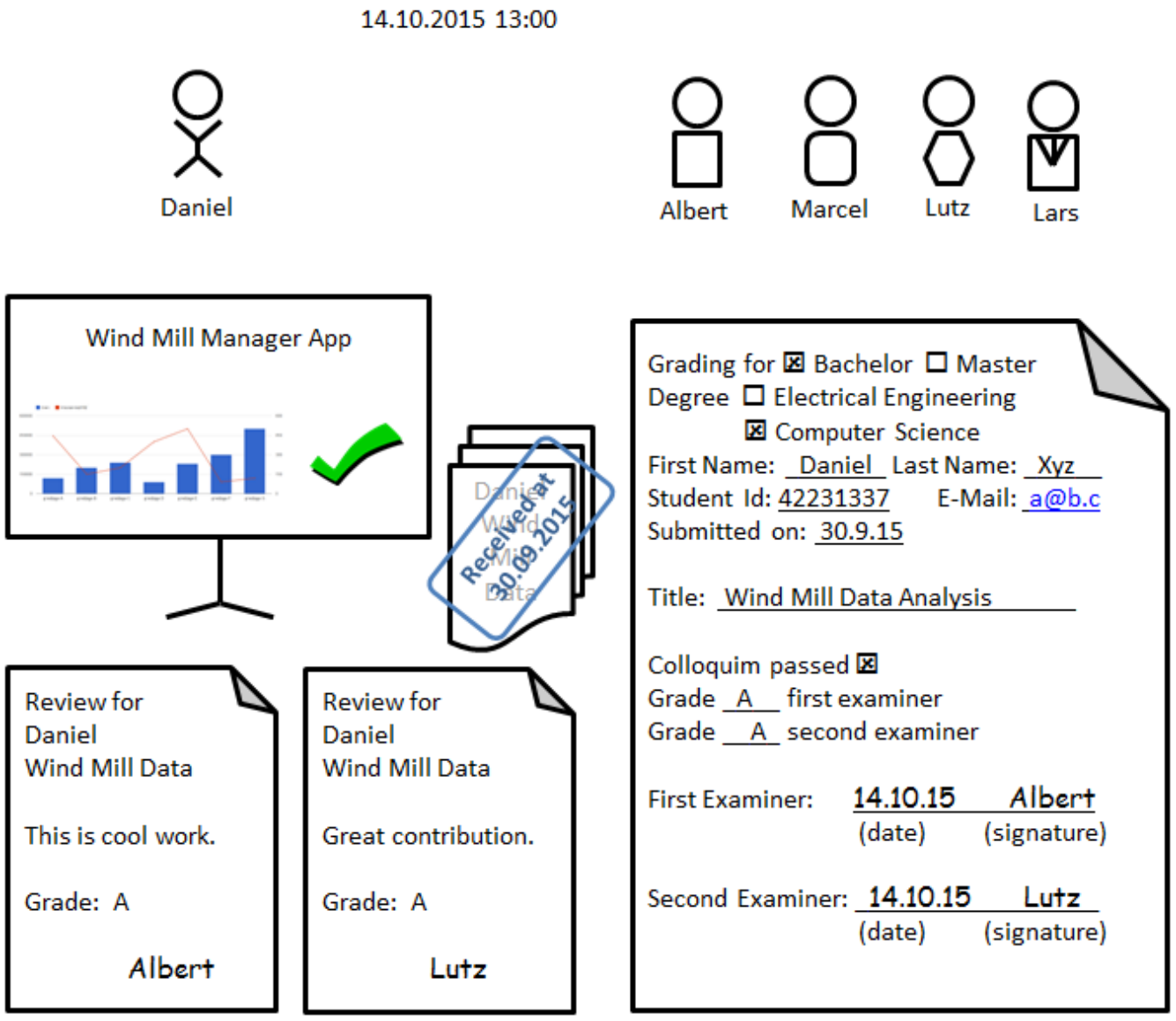


Figure 7: Software Story for Daniel's Bachelor Thesis, Colloquium

On October 14th Daniel had his Bachelor Thesis Colloquium, cf. Figure 7. Daniel gave a presentation of his results. There were several interested people from our group and especially Albert as first examiner, Marcel as university supervisor, Lutz as second examiner, and Lars as the industrial supervisor. After the presentation there were a lot of questions on details and a lively discussion. At the end, Albert, Marcel, Lutz, and Lars asked for some privacy and then discussed their impression from the work done by Daniel, from his thesis, and from his presentation. Finally, Albert and Lutz filled in and signed the Grading Form for Daniel's thesis. Now its almost done.

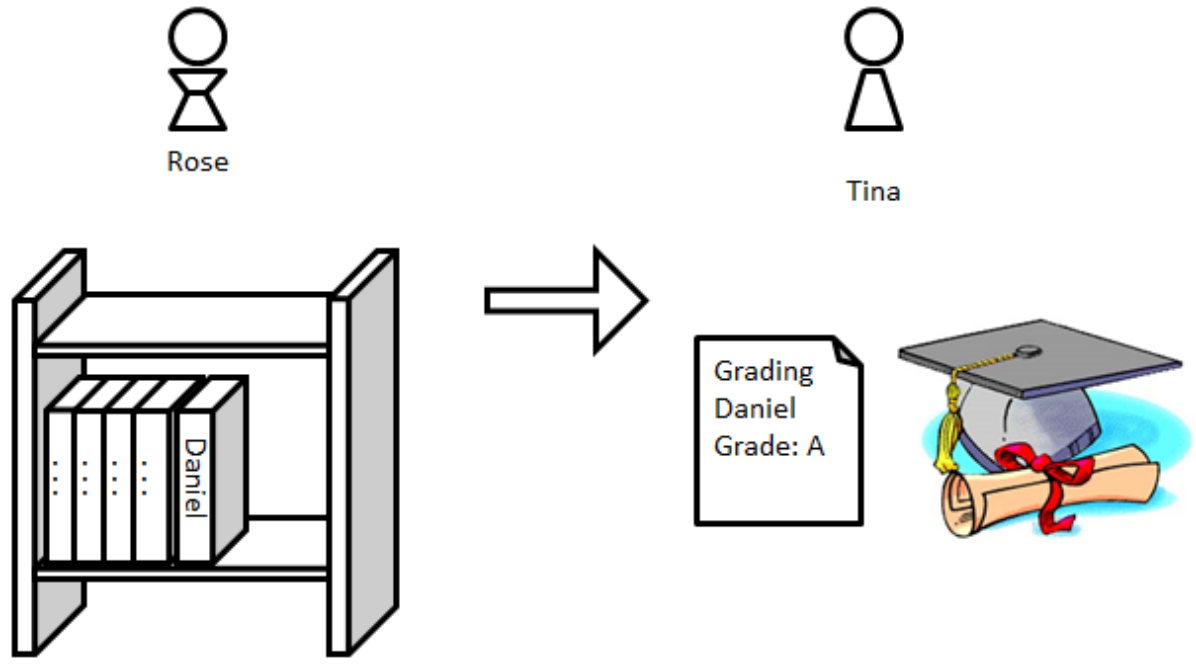


Figure 8: Software Story for Daniel's Bachelor Thesis, Diploma

As the very last administrative steps, Rose the secretary of our group adds the hard copy of Daniel's thesis to our library book shelf and she keeps a copy of the Grading Form for Daniel's thesis in our records, cf. Figure 8 left side. Then Rose sends the original Grading Form to Tina, the Student Service secretary. Tina, completes the Student Record of Daniel and creates Daniel's Diploma.

3 Example for Application Design

The Software Story of Daniel's Bachelor Thesis discussed in Section 3 has been written down by Albert as a first shot on the problem. It shows the current situation only. Later on it turned out that the story is still incomplete and that there are some other scenarios that are not covered by Daniel's story. For example, internal theses without an industrial partner are somewhat simpler as the coordination with the industrial partner can be omitted. In addition, cases were Albert is only second examiner require some internal steps beyond those performed by Lutz in Daniel's story.

As a first step towards the design of an application helping us to manage theses in our group we did a requirements elicitation workshop with Rose, our secretary, Tobi, the PhD student going to implement the thing, and Albert, eager to evaluate Software Stories in praxis. At the beginning of this requirements workshop, Albert explained Daniel's Software Story to Rose and Tobi using a large white board. During the discussion some missing elements popped up:

- For the very first step of our Software Story, we decided that we want a desktop application allowing us to record the protocol of the first meeting with a student bringing up the idea for a thesis. Thus we added a GUI mockup to the first step of Daniel's Software Story, cf. Figure 9.

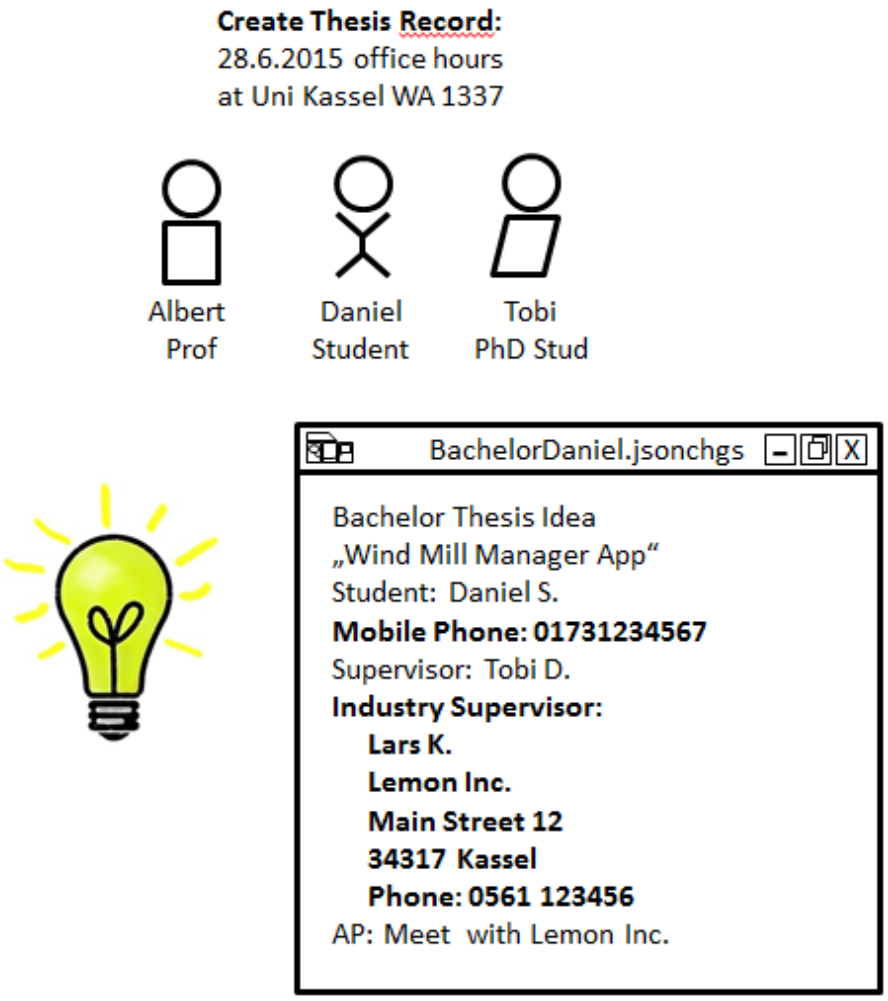


Figure 9: Gui Mockup Create Thesis

- Next, Tobi recalled that we were late for the first meeting at Lemon Inc. And we had no phone number nor contact name to call to announce that we are late. Similarly, there have been cases where we reached an enterprise in time but asked by the door man for our contact we could only name the student who had no entry in the enterprise's phone book. To avoid such problems, we decided that in case of an external thesis our new Thesis Management Tool shall force us to record the student's mobile phone number and the contact information for the industrial supervisor. See the bold parts of the GUI mockup in Figure 9.
- We also decided to give the first step the name "Create Thesis Record" shown in bold at the top of Figure 9.
- Then, we had a discussion whether the new Thesis Management Tool should also support the finding of dates for meetings. This usually involves checking Albert's Google Calendar and the calendars of the other participants and may require to set up a Doodle call. We anticipated problems in accessing the calenders of external participants and did not come to a conclusion for this feature. However, the Thesis Management Tool should allow to add action points or tasks on the fly and support us in keeping track of their execution.
- Beyond action points the new Thesis Management Tool might also allow to take notes on meetings. Thus, we would be able to protocol e.g. the outline of a thesis as done in Figure 2. However, the Thesis Management Tool was meant to facilitate the administration of theses and we were not sure whether it should deal with the actual content of theses. We decided to keep things simple at first and keep such a feature in mind for future extensions.
- As we reached the submission of Daniel's thesis in Figure 4, we noticed that Albert omitted an important step in the whole process: a thesis needs to be registered at the Student Service Office before you can submit it. Actually, the department rules require that you register your thesis before you start working on it. The rational behind this is that the amount of time spent on a e.g. Bachelor Thesis should not exceed 9 weeks. Thus, on registration the Student Service Office sets a deadline for the submission of the thesis and this deadline is controlled on submission. For some reasons students tend to be sloppy with the registration of their theses'. To enforce early registration, the Student Service Office set up the rule that one may not submit e.g a Bachelor Thesis earlier than 4.5 weeks after registering it. To address the

registration of Daniel's thesis, we added another step to our Software Story as shown in *Figure 10*.

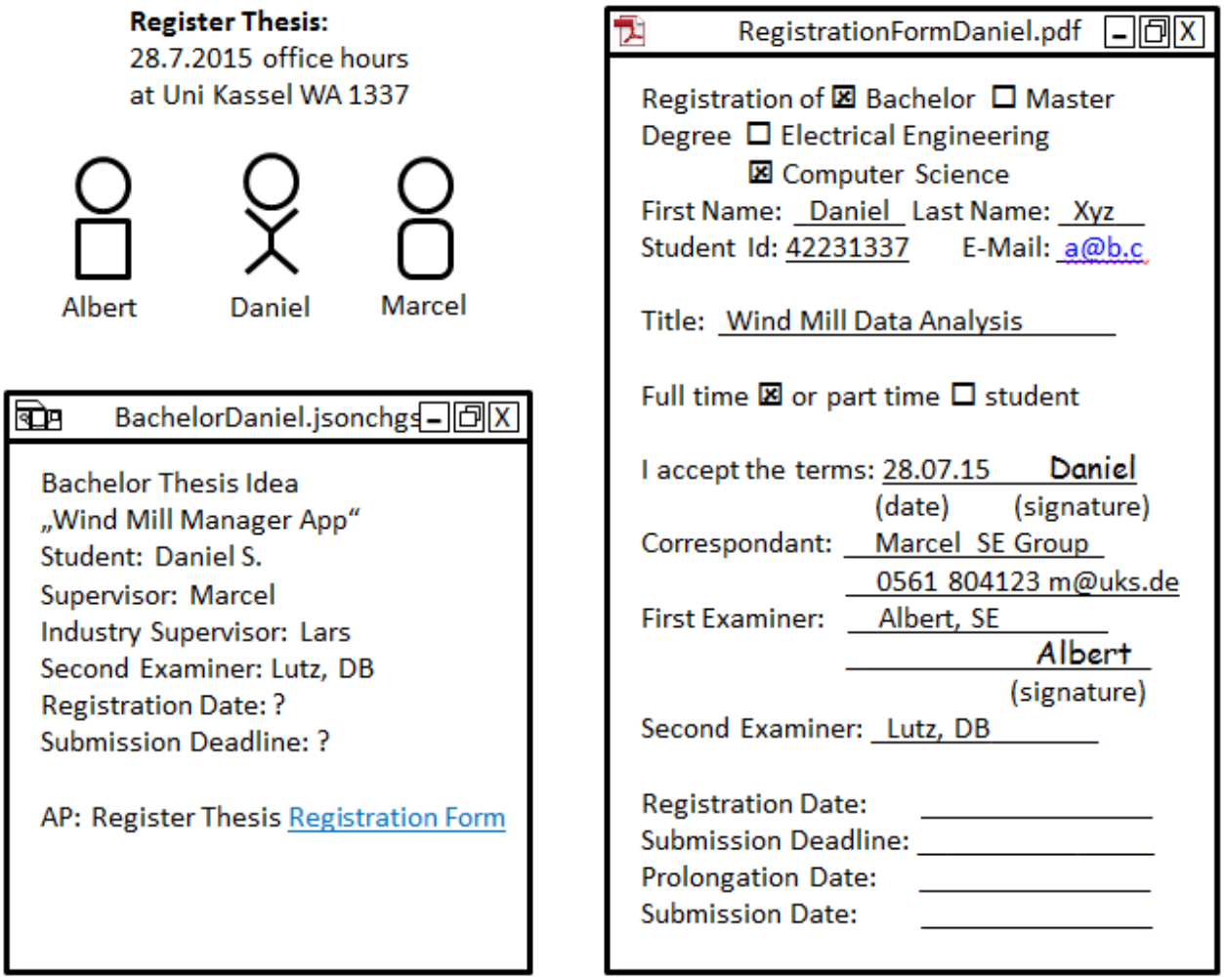


Figure 10: Thesis Registration

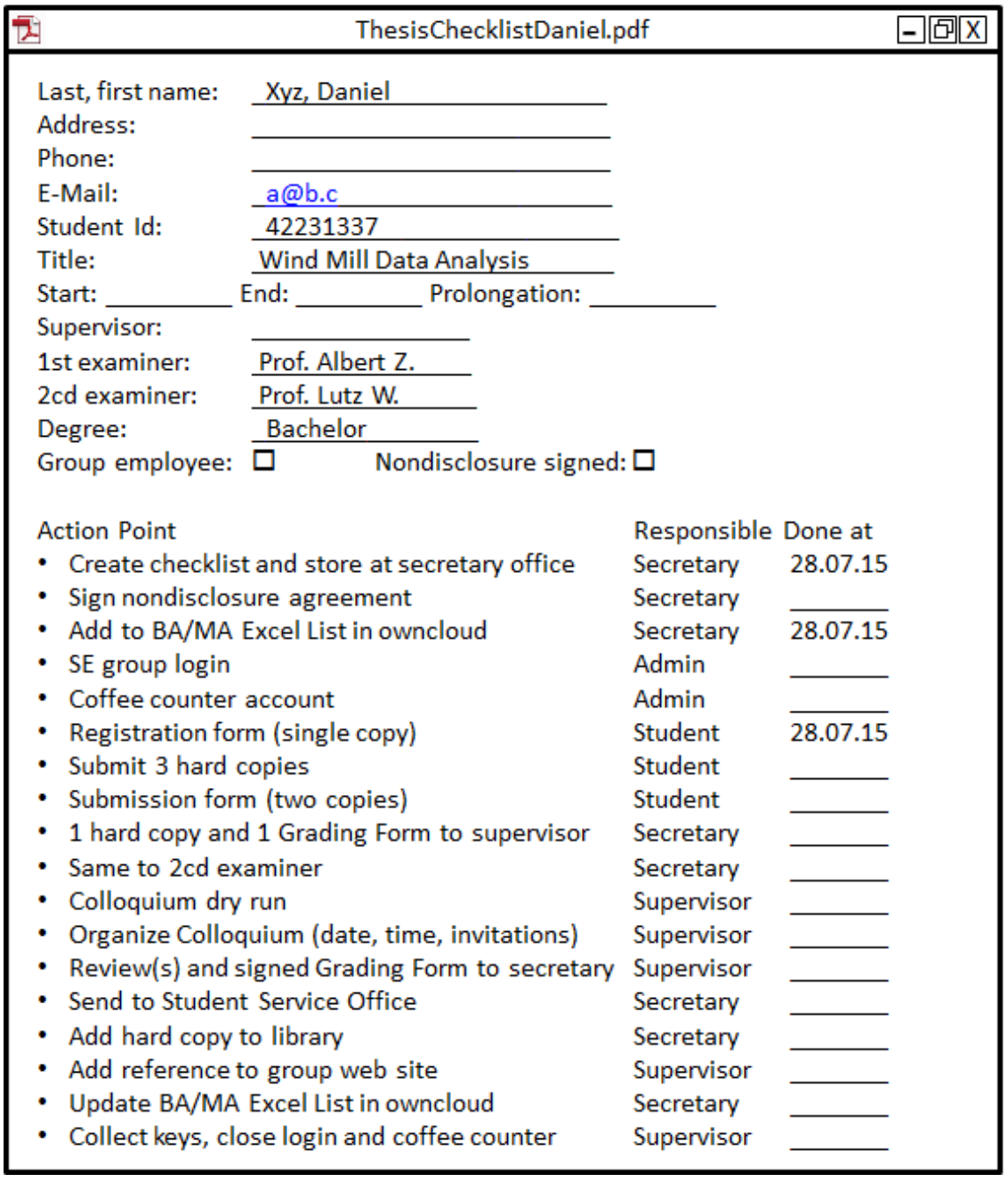
- Analysing the Register Thesis step brought up an essential idea for our Thesis Management Tool. We noticed that during the process one enters the credentials of various persons and e.g. the title of the thesis multiple times in multiple forms. Thus, the new Thesis Management Tool should have an address book keeping track of the contact data of all involved persons. Some contact data may also be imported from other sources, e.g. our group has an web based assignment management system for the courses given by us. Most likely, Daniel is already registered in this system with his credentials. Next, many information needs to be entered into our Thesis Management Tool and additionally in multiple PDF forms provided e.g. by the Student Service Office. Thus, it would be great if the new Thesis Management Tool provides links to the appropriate PDF forms and if the new Thesis Management Tool would be able to auto fill such forms with the data it already has. In the example of Figure 10 a click on the link provided by the Thesis Management Tool shown as mockup in the lower left should open some PDF tool with the Registration Form for theses and with as much fields already filled as possible. In our example, all shown information might be auto filled, except the signatures.

  This auto fill feature will greatly enhance the motivation of the members of our research group to actually use the Thesis Management Tool and to keep its content up to date. This is considered a killer feature.

- Next, our secretary Rose recognized that some students need to prolongate the submission deadline. Theoretically, this should not happen but sometimes reality strikes back. Our Student Service Office provides another PDF form for this case. Thus, our Thesis Management should allow to add such an action point on demand and it should auto fill that form too and keep track of the changed submission deadline.

- Finally, our secretary Rose came up with another PDF form which she uses to keep track of theses, cf. Figure 11. Our secretary's checklist revealed a number of new insights. First of all, we should have

asked her on the first run. She had much more insight in the administration of theses than the other group members. Second, there is a lot of redundancy in the current system. There are the forms used by the Student Service Office, the form used by our secretary, and an excel list in our owncloud, cf. Figure 12. And currently there is little motivation for most group members to enter the same information into multiple forms again and again. This results in the excel list being outdated all the time and in a lot of frustration for our secretary as she cannot answer questions from students nor questions from the Student Service Office like who is in charge of supervising a certain thesis and where are the hard copies of that thesis located and when the Colloquium is scheduled.



ThesisChecklistDaniel.pdf

Last, first name: Xyz, Daniel
Address: _____
Phone: _____
E-Mail: a@b.c
Student Id: 42231337
Title: Wind Mill Data Analysis
Start: _____ End: _____ Prolongation: _____
Supervisor: _____
1st examiner: Prof. Albert Z.
2cd examiner: Prof. Lutz W.
Degree: Bachelor
Group employee: ☐ Nondisclosure signed: ☐

Action Point	Responsible	Done at
• Create checklist and store at secretary office	Secretary	28.07.15
• Sign nondisclosure agreement	Secretary	_____
• Add to BA/MA Excel List in owncloud	Secretary	28.07.15
• SE group login	Admin	_____
• Coffee counter account	Admin	_____
• Registration form (single copy)	Student	28.07.15
• Submit 3 hard copies	Student	_____
• Submission form (two copies)	Student	_____
• 1 hard copy and 1 Grading Form to supervisor	Secretary	_____
• Same to 2cd examiner	Secretary	_____
• Colloquium dry run	Supervisor	_____
• Organize Colloquium (date, time, invitations)	Supervisor	_____
• Review(s) and signed Grading Form to secretary	Supervisor	_____
• Send to Student Service Office	Secretary	_____
• Add hard copy to library	Secretary	_____
• Add reference to group web site	Supervisor	_____
• Update BA/MA Excel List in owncloud	Secretary	_____
• Collect keys, close login and coffee counter	Supervisor	_____

Figure 11: Our secretary's check list

Bachelor /Master							
	Deadline	Colloquium	BA/MA	1.Examiner	2.Examiner	Supervisor	Remarks:
Ole	26.02.2015	End of April	BA	Prof. Zündorf	Prof. Sick	Tobi	does an internship in Austria
Alex	19.08.2015	Midth of Sept	MA	Prof. Zündorf	Prof. Stumme	Lennert	e-mail to Lennert 19.08.2015
Constantin	21.08.2015	End of Sept	BA	Prof. Zündorf	Prof. Wenzel	Tobias	
Seppel	02.09.2015	Midth of Okt.	BA	Prof. Zündorf	Prof. Geihs	?	
Chris	10.09.2015		MA	Prof. Zündorf	Prof. Gheis	Tobias	
Bob			BA	Prof. Zündorf	?	Lennert	still an idea
Mirco			BA	Prof. Zündorf	?	Marcel	
Ed	september 14?			Prof. Zündorf			
2. Examiner:							

Figure 12: Theses Overview in our Owncloud

- To overcome our problems with the administration of theses, we decided to build a workflow software that helps all of us to keep the informations about theses up-to-date and consistent. The users of this workflow software would be the scientific members of our group that supervise the theses and our secretary that deals with student requests and cummunicates with the people outside of our research group like the staff of the Student Service Office and superviser from other research groups. In addition, our secretary sends around a lot of copies of various documents. As GUI for the system we wanted some overview over all theses and their current states. This overview might be organized like the Excel table shown in Figure 12. In addition, for each single thesis we want a view like the checklist shown in Figure 11. This view should provide some common data about the student and the thesis at the top and it should show a checklist of todos and people in charge. Todo items that are done might go to the bottom of the list and form some kind of history. Current todo items should appear on top. Future todo items might show in the middle. Todo items that involve filling a PDF form shall provide links to that form and sould autofill it as much as possible.

As the new workflow software mimicks a checklist, we decided to name it E-Checkman for Electronic Checklist Manager.

As a next step we developed prototypes for the GUI of E-Checkman:

4 Notation Details

A Software Story consists of a sequence of Software Story Steps. Each step should have

- a name,
- a time and date,
- a location,
- some actors that execute the step or participate in the step,
- some picture or bullet list outlining the content of the step,
- some example data showing what information is processed and produced in this step.

The name of a Story Step frequently refers to a later implemenation of the illustrated activity. For example, in our E-Checkman system, the name "Register Thesis" of the Story Step of Figure 10 refers to an Action Point or todo item in the E-Checkman GUI that will be used to open the Thesis Registration Form of our department

and to (auto) fill this form. Thus, the name of a Story Step is merely used for tracing the described functionality in the later implementation.

Each Story Step should provide a time and date when this example step has been executed or will be executed. The first purpose of the example execution time is to help the people developing the Software Story to focus on the example level. It is not the "Register Thesis" dialog is opened by somebody but it is July 28th 10:42 and it is Albert together with Daniel and Marcel. We experienced that forcing domain experts and other Software Story developers to start with a conrete point in time helps them enormously to stick to a concrete example and to avoid to go to a more abstract rule level description.

Next, the concrete time and date helps to deal with parallel and concurrent actvities. At the rule level, concurrency issues require a lot of careful design of when a certain activity might be executed, which other activities need to be completed first and which set of activities might be mutual exclusive. In a concrete example, it is much easier to specify just when the activity is executed in this example. One may choose similar dates for two activities done by different actors in a single Software Story in order to give a hint that these two activities are independent from each other. One may also choose a sequence of points in time for a numbe of Story Steps in order to emphasize that the involved Story Steps trigger each other. One may also indicate that some Story Step happens e.g. several days later as another department or another organization is responsible for it and information has to be send around and the issue may need to wait before it is scheduled.

In general, timing and concurrency issues are complicated topics that need sound analysis. However, in early requirements engineering where domain experts are involved, it is hardly possible to address such issues in all details. Thus, we made the experience, that domain experts as well as software engineers can do a reasonable job in discussing concurrency issues just by providing time stamps for Story Steps.

The location where a Story Step is executed again helps the Software Story developers to focus on a concrete example. It also gives an idea of the kind of infrastructure you might expect for the execution of such a step. Story Step "Register Thesis" of Figure 10 is executed in the office of Albert at Kassel University. Thus, we can expect that a desktop computer is available and that we have access to the internet and with access to the E-Checkman system of our group. In other situations like on the right of Figure 1 we might only have a mobile system available or as shown in Figure 4, we might be in a different department and thus we might not be able to access e.g. E-Checkman.

Next, each Story Step should provide the actors that execute it or that are envolved. In our example, these actors are usually persons like Albert, Daniel, or Tina. These persons represent users of the described software system. However, to focus on the concrete example we use individual names to refer to the envolved actors and we also use individual icons for each of them. Still, the different Actors represent differnt roles or different kinds of users. These roles may be given as a second name or an a second name line like Albert Prof, Daniel Student or Tina Student Service. We frequently provide the role of an actor on its first occurence in our Software Story and omit it later on.

One additional remark on concrete named actors versus just role names: Different people doing the same job might frequently execute it quite differently. For example, some years ago the administration agent in charge for checking and refunding travel costs at Kassel University was Mister FourCorner. When Mr. FourCorner was in charge, it sufficed to fill some simple form and to put all your receipts in a bag and send it to him. If there was something unclear, he would phone you and clarify things, issue solved. The Mr. FourCorner retired. The new guy in charge was Mr. SpareTime. Mr. SpareTime did not call you back but he just send back your bag of receipts with a remark like "details are missing". This ususally requried quite a number of iterations and you were waiting for you money for ages. Finally, we had to change the process. Now you give your bag of receipts to Rose, our secreatary and she will ask you for the details and then send a detailed and complicated report to Mr. SpareTime. The lesson learned is, the same user role in the same process may function quite differently depending on the concrete person doing it. One might argue that such cases indicate missing process standardization and strictness. However, even with good standardization, in practice, different people do the same job differently. Thus, processes and the accompanying tools need to deal with different personalities and need to be flexibly adaptable to such issues. During requirments engineering this means, when developing a Software Story, knowing the concrete person who is doing a certain step will greatly facilitate to describe that step. It is much easier to describe how Tina handles the submission of Daniel's Thesis than to describe how some Student Service servant handles the submission of some student. To generalize from the concrete example to the rule level is the task of software developers, later on.

Our example Software Story models a workflow with several human actors. In other examples, some steps may be executed by a software system that e.g. does some consistency checking or that calculates a certain price or something like this. If an active piece of software is involved in a Story step, this active piece of software should be represented as an actor with some icon representing e.g. a robot or a computer or a cashier or any other piece of hardware. If your software already has a logo, you may also use that logo to represent it in a Software Story. A simple example might be adding the icon of a xerox machine to Figure 3.

The main content of a Story Step is some kind of powerpoint slide that outlines what is going on in this step. This outline may contain any pictures or drawings or bullet items. This content is mainly read by humans and has just the task to help the envolved domain experts and software engineers to get an idea of what is done in this step. On a white board, the participants will discuss the details of the Story Step content. When documented, some explaining text should be added to explain these things to the reader.

A very important part of the main Story Step content is the depiction of some example data. In our example Software Story, we depict several paper sheets or pdf forms showing notes, phone numbers, addresses, check boxes, text fields, etc. In Figure 9 we also mimic a screen dump showing an input form for some thesis and contact data. Such depiction of example data is extremely valuable for the requirements engineering task. Example data represented e.g. in forms that are familiar to the domain experts help those domain experts a lot to provide input to Software Stories and to explain which data they are dealing with. Similarly, the example data helps the software engineers to derive a data model for the desired software. The software engineers will most likely use some UML class diagrams to specify that data model for later implementation. While such class diagrams are very valuable for the software engineers and developers, domain experts usually do not understand class diagrams very well and they will not be able to spot faults in the class diagram design. Thus class diagrams will not help to clarify details and to resolve misunderstandings. Exmaple data shown to the domain experts in a familiar way does a much better job. Deriving the formal data model is done by the software engineers in a later step, easily. To our experience, example data contained in Software Stories is the most valuable part of a software story and thus we strongly suggest that Software Stories should always contain a lot of example data. Example data also helps you to find the right level of abstraction for your Software Story. If your Software Story does some kind of top down refinement, you may start with Software Stories like "Run SE Group Uni Kassel" containig Story Steps like "Administrate Theses" which are quite coarse grain. When you refine such complex Story Steps to more detailed activities, frequently the question arises whether we have reached a sufficient fine grained level of abstraction or whether we should still go on refining the steps. To our experience, as soon as you are able to give example data, you have reached the right level of abstraction.

5 Summary

Software Stories are a great means to do requirements engineering with domain experts, especially when those domain experts are not from the IT area. Focusing on concrete examples help the domain experts to explain and document their processes and by providing example data, the domain experts are able to provide great input for the data modelling step. To complement Software Stories during requirments engineering usecase diagrams may be used to group Software Stories and to give a general overview of the requirements. Thus usecase diagrams may help to structure the whole system while Software Stories explain the usecases in more detail. Thus, the usecases will help to structure the whole software development process and to group system functionality and to prioritize software development tasks. A thorough analysis of alternative scenarios may result in a larger number of Software Stories for a given System. Following the ideas of Story Driven Modeling [NJZ2013], Software Stories shall be turned into automatic (J)Unit [JUnit] tests. These Story tests serve two main purposes. First a Story Step ensures that the described functionality is actually implemented and working at least for the example scenarios. And second, the Story tests, help to ensure consistency accross multiple related Software Stories. Without such a consistency check, multiple Software Stories may easily contradict each other on how a certain step is done and why a certain decision is made and which example data is used and stored in a certain step. By turning Software Stories into JUnit tests, the software engineers will identify such inconsistencies and they may revisit the domain experts to resolve such issues. Once you have achieved a consistent set of JUnit tests for your Software Stories, these JUnit tests ensure the consistency and completeness of all your requirements. Note, your final system will need more tests than just the Story tests. You may also need a thorough system design using e.g. component and deployment diagrams and class diagrams. When designing algorithms, you may again use Software Stories or Storyboards for more fine

grained internal analysis. Such internal Software Stories may look much more technical and they may contain e.g. object diagrams or pseudo fragments. Overall, Software Stories are just a great help for requirements elicitation.

References

[BPMN] Object Management Group Business Process Model and Notation. http://www.bpmn.org/

[JUnit] Junit. http://junit.org/

[NJZ2013] Ulrich Norbisrath, Ruben Jubeh, Albert Zündorf. Story driven modeling. CreateSpace Independent Publ. Platform. ISBN-13: 978-1483949253. http://www.amazon.de/Story-Driven-Modeling-Ulrich-Norbisrath/dp/1483949257 2013

[Pohl2010] Klaus Pohl. Requirements Engineering: Fundamentals, Principles, and Techniques. Springer Publishing Company. ISBN:3642125778 9783642125775. 2010.