

# Teaching Students Collaborative Requirements Engineering: Case Study of *Red:Wire*

Dagmar Monett, Sven-Erik Kujat, Marvin Hartmann

**Abstract**—This paper discusses the use of a template-based approach for documenting high-quality requirements as part of course projects in an undergraduate Software Engineering course. In order to ease some of the Requirements Engineering activities that are performed when defining requirements by using the template, a new CASE tool, RED:WIRE, was first developed and later tested by students attending the course. Two questionnaires were conceived around a study that aims to analyze the new tool's learnability as well as other obtained results concerning its usability in particular and the Requirements Engineering skills developed by the students in general.

**Keywords**—CASE tool, collaborative learning, requirements engineering, undergraduate teaching.

## I. INTRODUCTION

**R**EQUIREMENTS Engineering (RE) is a sub-discipline of Software Engineering (SE) that includes all activities associated with “understanding a product’s necessary capabilities and attributes” [1] in order to define both its functionality and the constraints on its operation [2]. Wiegers and Beatty [1] split these activities into two groups: *Requirements development*, which encompasses the elicitation, analysis, specification, and validation of requirements; and *Requirements management*, which deals with tracking, managing, controlling, and tracing requirements.

As defined by the International Requirements Engineering Board e.V. (IREB) [3], RE is a “systematic [...] approach to the specification and management of requirements” that focuses on stakeholders’ desires and needs in a process-oriented way [4] that comprises the activities mentioned above. All of them should in one way or another be carefully considered, however. Both the scope and the depth of these RE activities depend on the software process model (e.g. waterfall, agile, etc.) that is used and how it is iterated. For example, documenting requirements in a waterfall approach such that they can be understood by all involved stakeholders and, at the same time, by following quality standards not only for single requirements but also for whole specifications could be a very complex task [5]. It is even a key activity in iterative and agile approaches. Documenting requirements is one of the most important activities in RE.

How to overcome the difficulties that might arise when documenting requirements has been widely reported in the

literature. See the guidelines for writing requirements in [1] and [6], to name a few. For example, Rupp and Wolf introduce the SOPHIST Set of *RE*gulations in [6], a compilation of rules to help requirements engineers “to systematically recognize and correct effects in natural-language requirements” in a proper way. Yet considering a set of guidelines or rules for documenting requirements might seem complicated at the beginning. Actually, it is highly dependent not only on well-founded knowledge about the topic or area in which requirements are to be defined but also on the experience the requirement engineers might have. This is why semi-structured methods, like template-based approaches, have been used as powerful mechanisms with the aim of simplifying and speeding up this RE activity. One of them will be introduced in what follows.

## II. SOPHIST TEMPLATE FOR SOFTWARE REQUIREMENTS

A template suggested by Rupp and Joppich [7] for creating high-quality requirements is shown in Fig. 1. It acts as a blueprint for phrasing requirements whilst avoiding some of the linguistic defects that could lead to loss or distortion of the information that is to be documented and, thus, to low-quality requirements.

The requirements template structures a requirement syntactically. Filling the template with content for each single requirement is a process that involves the following steps (see [7] for more), which are mainly performed in a sequential order:

- 1) Defining *the degree of legal obligation*: Setting the degree of importance that stakeholders assign to the requirement is of great significance. Keywords like “shall,” “should,” and “will” are often used to describe what is requested from the system.
- 2) Specifying *the functionality* that is requested: Clear “process verbs” help to distinguish what is to be done by the system.
- 3) Classifying *the type of functionality*: Is it an autonomous system action, a user interaction, or an interface requirement that depends on third parties?
- 4) Defining *the object and the adverbials*: Complementary information related to the functionality should be provided. This is commonly written at the end of the sentence.
- 5) Formulating *logical and temporal conditions*: A subordinate clause preceding the requirement sets down the conditions for the functionality to be executed.

D. Monett, Professor is with the Computer Science Dept., Faculty of Cooperative Studies, Berlin School of Economics and Law, Germany (e-mail: Dagmar.Monett-Diaz@hwr-berlin.de).

S.-V. Kujat, student and M. Hartmann, student are with the Computer Science Undergraduate Course IT2013, Faculty of Cooperative Studies, Berlin School of Economics and Law, Germany (e-mail: sv-kujat@t-online.de, marvin.hartmann1@gmail.com).

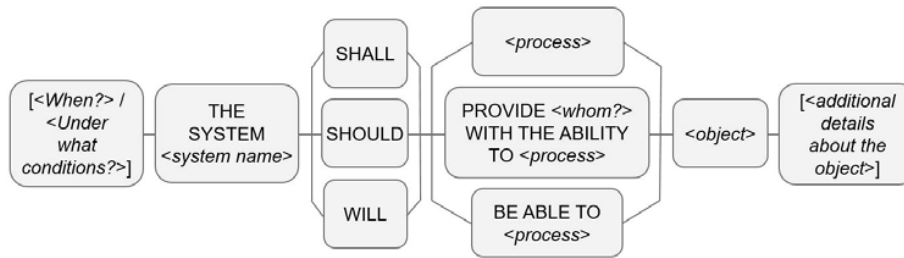


Fig. 1 SOPHIST requirements template, adapted from [7]

- 6) Applying the SOPHIST Set of *RE*gulations: A further revision of the requirement's semantic is advisable, since some linguistic defects could still be present.

However, mastering this process is not a trivial task for untrained requirements engineers or for RE novices, as undergraduate students attending SE courses might be in general. For this reason, as well as because there is enough related literature about it in the German language, training of Computer Science (CS) students at the Berlin School of Economics and Law (BSEL) on the documentation of requirements following the SOPHIST requirements template has been an important learning goal.

#### A. Educational Applications of the Template

The SOPHIST requirements template has been used in numerous software industry projects so far [8]. It has also been used successfully in Software Engineering courses at BSEL since 2011. In particular, the SOPHIST requirements template has found application in the following educational topics:

- *Natural language description of the flow in UML<sup>1</sup> activity diagrams:* UML activity diagrams are given to the students in SE lab sessions. The students describe the diagrams' flow in a collaborative exercise using, for example, Google Docs. The sentences should have the same syntactic structure as the SOPHIST requirements template. A discussion follows, where learners and instructors revise the answers and apply the SOPHIST Set of *RE*gulations further, if needed.
- *Construction of UML activity diagrams from requirements that already follow the SOPHIST requirements template:* Documented requirements in the form of sentences that follow the template from Fig. 1 are given to the students. They construct UML activity diagrams using a UML tool. Notation elements and alternative diagrams are discussed with the instructors after the students have presented their solutions.
- *Documentation of software requirements in SE course projects:* Students work on their course projects in class and remotely. They document the software requirements using, for example, Google Sheets and Google Docs. Instructors give feedback on the state of their work

by giving advice on the correct use of the SOPHIST requirements template, when necessary.

Not only the theory but also some interesting examples and exercises as well as suggestions on the use of the SOPHIST requirements template can be found in [9], [10]. Having followed many of them on the SE courses at BSEL, we have found more and more students who successfully use the syntactic structure of the template to define requirements of course projects in other CS modules and in their own Bachelor theses at the end of their studies at BSEL. This speaks for itself and gives us strong positive feedback about our educational approach.

#### B. CASE Tools for RE

Computer-Aided Software Engineering (CASE) tools have been used to support almost all processes and activities of the software life-cycle for decades. There exist many CASE tools for supporting RE and its activities (an updated list of requirements management tools can be found in [11]). A first analysis of several tools for RE tasks provided no results for our concrete educational purposes; however, available CASE tools for requirements documentation and management were either commercial or offered no support for the SOPHIST requirements template at all.

Our goals were fairly simple: We needed a CASE tool (see [12] for a classification of CASE technology) to support specific tasks from the software life-cycle. In particular, we needed a software product for supporting the activities of requirements development that could be used in the tasks of documenting and analyzing software requirements using the SOPHIST requirements template in students' course projects. We wanted the students to focus on the definition of highly-qualitative requirements by using the template-based approach from Rupp and Joppich [7] without having to define all template components from scratch or by hand.

In consequence, the decision to implement a simple, easy to use CASE tool for use by CS students of the SE courses at BSEL was an alternative to automating the process of defining requirements using the SOPHIST requirements template introduced so far. It should have a very short learning curve, however, since many researches have indicated not only the benefits of CASE tools but also their learning curve problems and learnability issues in educational settings (see [13] for a deeper discussion on these subjects).

<sup>1</sup>Unified Modeling Language.

The implementation of such a new tool was considered as a topic for undergraduate student research projects in advanced semesters. The first two prototypes were developed by two CS students after they had attended their respective SE undergraduate courses from fall 2012 and fall 2013 (one student each time). In both cases, the students had already mastered the work with the template-based approach in class and, thus, it was much easier for them not only to define the requirements for the new software but also to design and implement it. A third prototype was developed by two other students between the months of January and July 2015 after they had attended the SE course from fall 2014. That prototype benefited greatly from the former versions and added core functionalities for team work. It will be introduced in the next section.

### III. RED:WIRE

RED:WIRE is both a documentation and a requirements management tool, developed by and for students, that supports the definition and management of user and software requirements, as suggested in [14], and mirrors the template-based definition of requirements from [7] in a collaborative way. It allows students in particular and users in general to fill out a pre-defined template that structures a requirement syntactically and that avoids, to a great extent, the presence of linguistic defects that are common in the natural language. The most important issues concerning its development will be introduced in the sections that follow.

#### A. Analysis and Design

RED:WIRE's graphical user interface was designed using the Bootstrap (see <http://getbootstrap.com> for more) CSS<sup>2</sup>-framework. The framework allows an easy implementation of the basic layout structure, which is separated into four major areas: header, footer, menu, and content areas.

Figs. 2-4 show screenshots of RED:WIRE's dashboard. The header is located at the top of the page and contains the RED:WIRE logo. The footer is located at the bottom of the page and holds the changelog, the current version, and contact information about the developer team.

The menu is located at the left hand side and is a quarter of the width of the document in size. Below the menu is a news feed with changes and alerts about actions performed. Finally, on the right hand side and three quarters of the width of the document in size, there is a content area that changes when the user requests a server-side action. It displays the results afterwards and is the area that has the most interactions of RED:WIRE's users.

Fig. 2 shows RED:WIRE's functionality for defining software requirements. It can be accessed by clicking on the CREATE REQUIREMENT option from the menu. In its current implementation, it uses a German version of the template-based approach from Rupp and colleagues introduced

in Section II. The template's components are set in a fixed position to help users enter the requirement. Some additional components were added to the template, too: an identifier to uniquely differentiate the requirement, information about dependencies or other requirements directly related to the current one that is being defined, a priority, and the status of the requirement (i.e., *in backlog*, *in progress*, *being tested*, *done*) in order to track changes to the requirement and manage it better.

Fig. 3 shows the same area where the extended SOPHIST requirements template is used but filled with content, that is, for the case of one specific requirement. Fig. 4 shows a list of requirements that have been defined in this way. The user can not only edit previous information but also remove requirements from the list, in case they are no longer needed. The table with requirements can be accessed too by clicking on the HOME option from the menu. There is also an option to export the list of requirements to other formats.

Fig. 5 shows the login page that appears when the Web server is accessed. It has a lightweight design consisting only of two input forms, in which the user's credentials are inserted. A dialog pops up when a user clicks on the REGISTER button. It contains simple input forms for the user's name, password, and e-mail address.

There is an administrator page, too, which has both the same structure and the same design as the dashboard page but fewer options. On its right hand side, it shows a list of all registered users. The administrator can remove users from the database if needed.

#### B. Implementation

The implementation of RED:WIRE took place in a client-server architecture. The backend-server consists of an Apache web server that serves the HTML pages to the clients, a MySQL database to store all the software and user data, and additional PHP<sup>3</sup> modules for the Apache web server to interpret the clients' requests. These are sent using JQuery's AJAX,<sup>4</sup> which calls PHP files. Fig. 6 shows the scheme of the relational database structure that is used in RED:WIRE. It consists of four tables for dealing with users who are part of teams and who define requirements, which can be of different types or categories (e.g., functional and non-functional requirements).

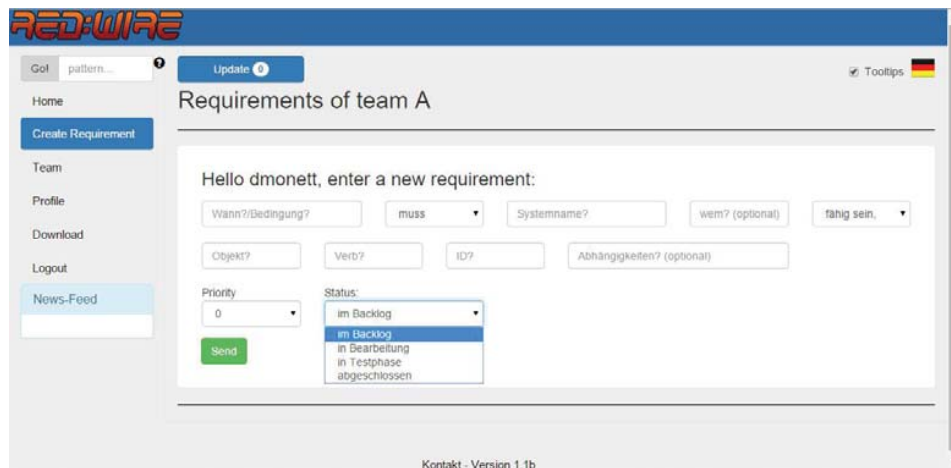
The frontend is written in HTML.<sup>5</sup> The HTML elements are formatted by CSS. The responsive layout uses CSS-classes of the Bootstrap framework to get a dynamically scalable page that also works on a mobile device. Another framework, called SASS (see more at <http://sass-lang.com>), is used to make the CSS development for different devices easier. The animations and function calls are generated by JavaScript, explicitly JQuery (available at <https://jquery.com>), to ease the manipulation of HTML data object models and to use asynchronous calls to access stored data via PHP files. By

<sup>2</sup>Cascadian Style Sheet.

<sup>3</sup>Pre-Hypertext Processor.

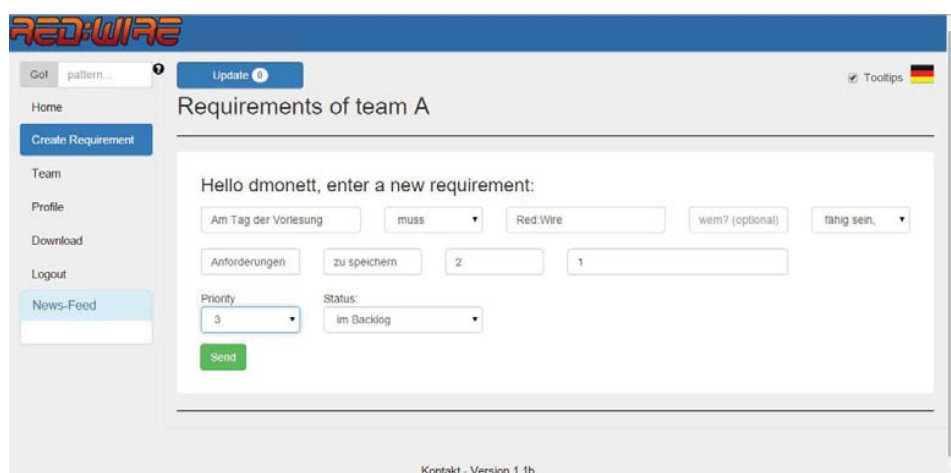
<sup>4</sup>Asynchronous JavaScript and XML.

<sup>5</sup>Hypertext Markup Language.



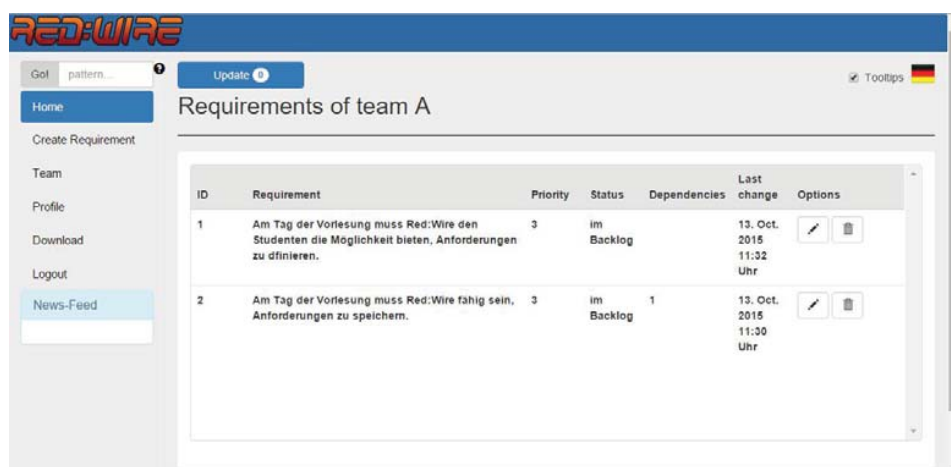
The screenshot shows the 'Requirements of team A' window in Red:Wire. The left sidebar contains links: Home, Create Requirement, Team, Profile, Download, Logout, and News-Feed. The main area has a header 'Requirements of team A' and a sub-header 'Hello dmonett, enter a new requirement:'. Below this is a form with several input fields: 'Wann?/Bedingung?' (containing 'Am Tag der Vorlesung'), 'muss' (a dropdown), 'Systemname?' (containing 'Red:Wire'), 'wem? (optional)', 'fähig sein.' (a dropdown), 'Objekt?', 'Verb?', 'ID?' (containing '2'), 'Abhängigkeiten? (optional)' (containing '1'), 'Priority' (containing '3'), and 'Status' (a dropdown with 'im Backlog' selected). A green 'Send' button is at the bottom left. The footer says 'Kontakt - Version 1.1b'.

Fig. 2 Red:Wire's requirements template window without content (German version)



This screenshot is identical to Fig. 2, showing the 'Requirements of team A' window with the same form fields and values as described in the previous figure.

Fig. 3 Red:Wire's requirements template window filled with content. Translated to the English language, the requirement reads: "On the day of the lecture, Red:Wire shall save requirements." Id=2, Dependencies=1, Priority=3, Status=*in backlog*



The screenshot shows the 'Requirements of team A' window with a table listing two requirements. The table has columns: ID, Requirement, Priority, Status, Dependencies, Last change, and Options. Requirement 1 is 'Am Tag der Vorlesung muss Red:Wire den Studenten die Möglichkeit bieten, Anforderungen zu definieren.' with ID 1, Priority 3, Status 'im Backlog', and no dependencies. Requirement 2 is 'Am Tag der Vorlesung muss Red:Wire fähig sein, Anforderungen zu speichern.' with ID 2, Priority 3, Status 'im Backlog', and 1 dependency (on requirement 1). Both were last changed on 13. Oct. 2015.





ID	Requirement	Priority	Status	Dependencies	Last change	Options
1	Am Tag der Vorlesung muss Red:Wire den Studenten die Möglichkeit bieten, Anforderungen zu definieren.	3	im Backlog		13. Oct. 2015 11:32 Uhr	 
2	Am Tag der Vorlesung muss Red:Wire fähig sein, Anforderungen zu speichern.	3	im Backlog	1	13. Oct. 2015 11:30 Uhr	 

Fig. 4 A list with two requirements and their dependencies. The second requirement depends on the realization of the first one



doing this, the data become available shortly after the call's execution, without reloading the page.



Fig. 5 Red:Wire's login window (English version)

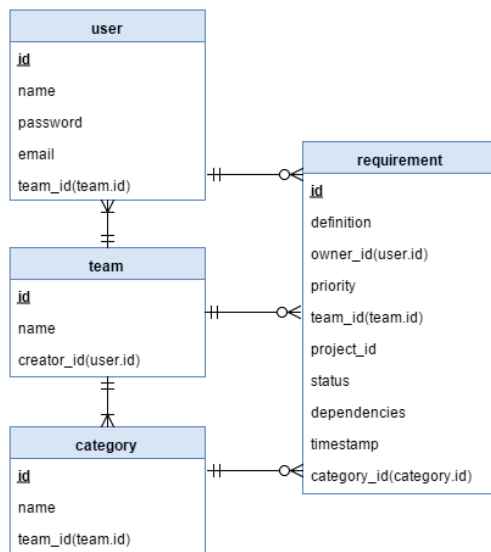


Fig. 6 Database tables and their relationships

### C. Testing

Different test scenarios were defined for black-box testing RED:WIRE's functionalities. They mainly comprised both functional and usability tests of menu options and end users' activities on different Web browsers (Internet Explorer, Mozilla Firefox, and Google Chrome) like the following ones:

- Logging in, registering and logging out users.
- Database and server connectivity and data exchange.
- Management of teams with several users.
- Working with requirements and lists of requirements.
- Performance when more than a thousand requirements are managed.
- Changing the profile data of users.

The source code was also tested following white-box testing methods. A deeper usability testing was performed with a group of students from the SE course from fall 2015. More on this will be addressed in the next sections.

## IV. RESEARCH METHODOLOGY

Students from the SE course from fall 2015 were informed about the research purposes of our study, that is, related to the use, testing, and evaluation of the new CASE tool developed by former students in particular, that is, RED:WIRE, as well as the learning and mastering of RE activities and processes in general. Appropriate closed-ended questions with Likert scales as well as a few open-ended questions were prepared for inclusion in two surveys: a pre-survey, before students' work with requirements in class or on their course projects, and a post-survey, by the end of term and after the final project presentations and work with the new CASE tool. The use of RED:WIRE was mandated for documenting and managing requirements. Other tools for collaborative work, for example, were suggested by the teaching staff but the students had free choice regarding their use.

The students were advised on the concrete use of RED:WIRE as part of a laboratory session. First, a detailed example of how to describe the flow of a UML activity diagram using the SOPHIST requirements template (see Section II-A) was demonstrated and discussed in class. Then, each student had the opportunity to write the natural language description of some actions and decision nodes of a new UML activity diagram that was distributed to them, thereby using the syntactic structure of the SOPHIST requirements template. They documented the descriptions collaboratively using Google Docs. A discussion in plenum followed, where the different activity flows were analyzed and complemented with further input from students. Finally, each student documented her or his resulting sentences in RED:WIRE, where the template was already available to ease the process.

Further, a meeting with each team and with the presence of the teaching staff was coordinated. In these meetings, team leaders discussed project tasks with the team members and distributed tasks among them. The teaching staff gave advice on the different major areas into which the course project could be divided (e.g., software requirements for each of the course project subjects). The teams agreed on the selection of experts for each of these subjects as well as setting initial deadlines for the RE activities to be performed by them.

Some lectures and laboratory sessions included extra time for teamwork activities and exercises. For example, further methods and tools were introduced and explicitly used for modeling and graphically documenting requirements (like UML tools) in class. Simultaneously, a close contact with RED:WIRE's developers was maintained because they fixed eventual errors while working on new versions of the program.

### A. Research Questions

One of the main goals of our study was for several students in the role of end users to provide valuable feedback concerning functional and non-functional aspects of the new CASE tool, that is, RED:WIRE. The chance for an entire undergraduate course to test a tool that is still under development by other undergraduate students on more

advanced courses is an excellent opportunity. Furthermore, RED:WIRE is expected to successfully support requirements analysis tasks by helping students with the definition of high-quality requirements. Similarly, collaboration between team members for documenting and managing requirements when using RED:WIRE is expected to be fostered. Since the SE course places emphasis on RE processes and their tasks, it is also expected that students will develop new skills in these topics, including the learning of new CASE tools. Thus, the main research questions driving our investigation are as follows:

- Q1:** Do students improve their RE skills (perceived improvement) after attending the course?
- Q2:** Can RED:WIRE be learned easily by the students?
- Q3:** Does RED:WIRE allow the successful documentation and management of requirements?
- Q4:** Is RED:WIRE an adequate CASE tool for collaborative work when documenting requirements?
- Q5:** Are students satisfied with the use of RED:WIRE (subjective satisfaction)?
- Q6:** Are there functional and/or non-functional issues that could be fixed after testing RED:WIRE?
- Q7:** Are there functionalities that are new or nice to have that could be added to RED:WIRE's repertoire?

The following sections present the two surveys that were conducted together with the results that were obtained.

#### B. Pre-Survey Results

A short pre-survey was administered to the students after the RE lectures in which the theoretical content was introduced and before the corresponding practical sessions took place in the laboratory. Of the 33 students attending the SE course, a total of 23 answered the pre-survey questions, giving a response rate of 70%. There was one female student for every ten male students in the group. The gender and age proportions of those who answered the pre-survey were as follows: 13% of the respondents (3 out of 23 students) were female and 87% were male; 13% were aged 19 years or less, 13% were aged 26 years or more, and the rest were between 20 and 25 years old.

One of the goals of the pre-survey was to find out whether the students had had any experience with software requirements before attending the course. The question was answered positively by 48% of the respondents (11 students), that is, almost half of the students had worked with software requirements to some extent. However, less than half from them had any experience with eliciting, documenting, specifying, or managing software requirements. With respect to a question asking about online tools for documenting requirements, only 34.8% of the participants (8 students) answered that they knew at least one tool for doing this. Among them, only three students (13% of the total) had actually used an online tool for this RE task.

How the above records changed after the students presented the results of their course projects at the end of the semester will be topic of the following section.

#### C. Post-Survey Results

A similar but extended survey was administered to the students at the end of the SE course. It included a subset of the questions from the pre-survey, to allow comparison, as well as new questions to analyze the usability in general and the learnability in particular of RED:WIRE. The students had a free choice on the selection of software tools to support teamwork on their course projects. However, the use of RED:WIRE was mandatory for documenting requirements, for the reasons introduced in Section IV.

A total of 24 students (one more than in the pre-survey) answered the post-survey questions, giving a response rate of 72.7%. Two of them (8.3%) were female and the rest (91.7%) male; 25% of the respondents were aged 19 years or less, 58.3% were aged between 20 and 25 years, and 16.7% were aged 26 years or more.

Two thirds of the respondents (16 students) gave a positive answer to the question involving experience with software requirements. This represents an increment of about 19% with respect to the pre-survey responses. The great majority (95.8%) answered that they had had experience with eliciting (58.3%), documenting (58.3%), specifying (37.5%), and managing (16.7%) software requirements. This shows that RE activities were assigned differently in the teams and that not every student was equally involved in all of the tasks of the RE development, contrary to what we initially supposed.

The work of Senapathi [13] was particularly helpful in the selection of questions to be posed to the students to evaluate the learnability of RED:WIRE. The questions are listed in Table I.

TABLE I  
CLOSED-ENDED LEARNABILITY QUESTIONS ORGANIZED BY  
CATEGORIES, ADAPTED FROM [13]

Ease of learning	(A) It was easy for me to get started and to learn how to use RED:WIRE.
	(B) I was able to use RED:WIRE right from the beginning, without having to ask other persons for help.
	(C) I was able to try out new RED:WIRE functions by myself.
	(D) It was easy for me to remember commands from one session to another.
	(E) The explanations, tooltips, and headings that were provided helped me to become more and more skilled at using RED:WIRE.
Consistency	(F) RED:WIRE is consistently designed, thus making it easier for me to do my work.
	(G) I find that the same function keys are used throughout the program for the same functions.
Predictability	(H) RED:WIRE behaves similarly and predictably in similar situations.
	(I) When executing functions, I get results that are predictable.
Error messages	(J) If I make a mistake while performing a task, I can easily undo the last operation.
	(K) Error messages clarify the problem.
	(L) I perceive the error messages as helpful.

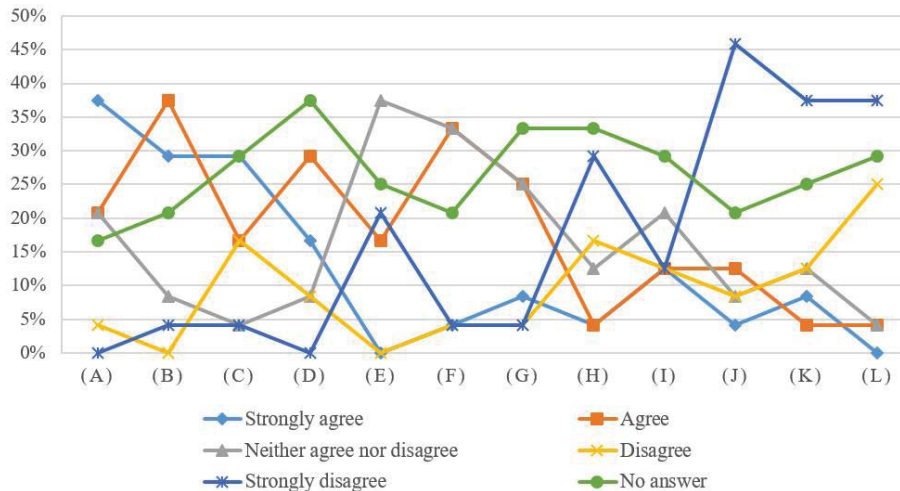


Fig. 7 Learnability of Red:Wire according to students' responses

Fig. 7 shows the responses that were obtained for each learnability question from (A) to (L). The most distinctive results for the different categories can be summarized as follows:<sup>6</sup>

#### *Ease of learning*

(A) The majority of the respondents (58.3%) agreed or strongly agreed that it was easy to get started with RED:WIRE and to learn to use it, (B) 66.7% were able to use the CASE tool without requiring help from third parties, (C) 45.8% were able to try out new functions without any problem, and (D) 45.8% responded that they could easily remember RED:WIRE's commands from one section to another. However, (E) 37.5% of the respondents were not sure whether explanations, tooltips, and headings had a positive or negative impact on helping them becoming more skilled when using the tool. The explanations were viewed positively by 16.7% of them and negatively by 20.8%.

#### *Consistency*

(F) Many more respondents agreed with the statement that RED:WIRE was consistently designed, thereby helping them to do their work (37.5% agreed or strongly agreed), than agreed with the contrary statement (8.33% disagreed or strongly disagreed). (G) The responses for the same function keys being used throughout the program for the same functions show very similar results.

#### *Predictability*

(H) RED:WIRE's behavior cannot be predicted positively in similar situations. At least 45.8% of the respondents thought that this was the case after

having to deal with software errors or unexpected functioning. (I) In general, prediction of the results of RED:WIRE's behavior is very unstable at the moment (25% positive answers, 20.8% neutral, and 25% negative ones).

#### *Error messages*

(J) For 54.2% of the respondents, undoing the most recent operation was not always possible in RED:WIRE; (K) 50% found error messages confusing, and (L) 62.5% found them not helpful at all.

The above overall ratings confirm both the results obtained by Senapathi and the ones from the literature discussed by her in [13]. Essentially, the highest rated questions, that is, the ones with the most favorable evaluations, belong to the category *ease of learning*, which indicates that RED:WIRE could be a good option for novices when documenting requirements (i.e., it has a short learning curve and is easy to learn). That the lower rated questions belong to the category *error messages* is not surprising and should not be correlated with other categories, as discussed in [13], too.

Other general and usability questions were aimed at finding out about the overall subjective satisfaction with RED:WIRE, its performance, and the collaborative work with the tool, to name but a few. Table II shows these questions and their responses.

The ratings from Table II show a clear need for further development and testing of RED:WIRE. At the same time, the feedback provided by the students is very helpful in this respect and will be considered by the current developers of RED:WIRE as well as by new students that join its team. Interestingly, students rate their software requirements analysis capabilities without CASE tool support very favorably. This points to the successful achievement of the learning goals that were set in the lectures and laboratory sessions in advance.

<sup>6</sup>On average, 26.7% of the respondents provided no answer to each of these questions. This is related to the dissimilar use of RED:WIRE and to the distribution of RE tasks in the teams: not all team members worked with the tool, as originally advised and supposed.

TABLE II

OTHER GENERAL CLOSED-ENDED QUESTIONS AND THEIR RESPONSES (MEAN AND STANDARD DEVIATION) FOR RESPECTIVE 5-POINT, UNMARKED SEMANTIC DIFFERENTIAL SCALES (WITH 1 POINT FOR NEGATIVE TO 5 POINTS FOR POSITIVE WORDS)

Question	Mean	Std.dev.
(i) How do you assess your software requirements analysis capabilities without a CASE tool support?	<b>3.58</b>	0.76
(ii) How do you assess your software requirements analysis capabilities with the support of RED:WIRE?	2.83	1.28
(iii) How satisfied are you with software requirements documentation functionalities of RED:WIRE?	2.04	1.06
(iv) How satisfied are you with the performance of RED:WIRE?	2.46	1.12
(v) How appealing do you find the graphical user interface of RED:WIRE?	2.50	0.96
(vi) How satisfied are you with collaborative work functionalities in RED:WIRE?	2.17	1.14
(vii) How much did RED:WIRE support you with the software requirements documentation?	1.92	0.86
(viii) How much you have improved your software requirements skills by working with RED:WIRE?	<b>1.88</b>	0.78

## V. CONCLUSIONS

RE processes deal with the understanding of products' capabilities and attributes by involving stakeholders in several activities and tasks that aim to produce high-quality requirements. When supported by appropriate CASE tools, these activities can be achieved better and stakeholders' interactions and work can both be eased and improved. Teaching and training of novices on these topics require a careful selection of contents and tools together with adequate learning goals that seek to develop and enhance the required RE skills. Some of these skills include the documentation and management of requirements. This paper introduced the approach we used to achieve them in an ES undergraduate course from fall 2015.

One distinctive aspect of our approach was the use of a new CASE tool, RED:WIRE, developed by other undergraduate students from more advanced years. We expected that the tool would be easy to learn and conceived a research study focusing on research questions presented in two different questionnaires. Some of our expectations were fulfilled. Others gave us a starting point for further work.

In general, students perceived that they had improved their RE skills after attending the course. Laboratory sessions and explicit demonstrations of how high-quality requirements can be documented contributed to that. The mandated CASE tool can be easily learned by non-experts and was indeed used by all the teams. However, its functionalities could not be exploited completely mainly due to unexpected behaviors or lack of maturity as a software application. Finally, through administering the questionnaires, it was possible to gather not only very useful feedback from the end users (also testers) but also practical input on which new capabilities and attributes this software product should include in the future.

## REFERENCES

- [1] K. Wiegers and J. Beatty, *Software Requirements*, 3rd ed. Redmond, Washington: Microsoft Press, 2014.
- [2] I. Sommerville, *Software Engineering*, 9th ed. Pearson, Addison-Wesley, 2011.
- [3] IREB. International requirements engineering board. International Requirements Engineering Board. Accessed: Jan. 14, 2016. [Online]. Available: <https://www.ireb.org/en>
- [4] M. Glinz, *A Glossary of Requirements Engineering Terminology*, IREB e.V. and Department of Informatics, University of Zurich, May 2014.
- [5] C. Rupp, "Chapter 1: In medias RE," in *Requirements-Engineering und Management: Aus der Praxis von klassisch bis agil*. Hanser, 2011, pp. 1–21.
- [6] C. Rupp and E. Wolf, "Chapter 6: The SOPHIST Set of REgulations – Psychotherapy for Requirements," in *Requirements-Engineering und Management: Aus der Praxis von klassisch bis agil*. Hanser, 2011, pp. 115–156.
- [7] C. Rupp and R. Joppich, "Chapter 7: Templates – Construction Plans for Requirements and for More," in *Requirements-Engineering und Management: Aus der Praxis von klassisch bis agil*. Hanser, 2011, pp. 157–176.
- [8] SOPHIST. Sophist: Start. Accessed: Jan. 14, 2016. [Online]. Available: <https://www.sophist.de/en/start/>
- [9] S. Kleuker, *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten*, 2nd ed. Munich: Vieweg+Teubner Verlag, 2011.
- [10] C. Rupp, *Requirements-Engineering und Management: Aus der Praxis von klassisch bis agil*, 6th ed. Munich: Hanser, 2014.
- [11] A. Birk and G. Heller. (2015) List of requirements management tools. Accessed: Jan. 14, 2016. [Online]. Available: <http://makingofsoftware.com/resources/list-of-rm-tools>
- [12] A. Fuggetta, "A Classification of CASE Technology," *IEEE Computer*, vol. 26, no. 12, pp. 25–38, 1993.
- [13] M. Senapathi, "A Framework for the Evaluation of CASE Tool Learnability in Educational Environments," *Journal of Information Technology Education*, vol. 4, pp. 61–84, 2005.
- [14] C. Mazza et al., *Guide to the selection and use of CASE tools*, ESA Board for Software Standardisation and Control, European Space Agency, May 1994.