STAGE – A Software Tool for Automatic Grading of Testing Exercises

Case Study Paper

Sebastian Pape
Chair of Mobile Business & Multilateral Security
Goethe University Frankfurt
sebastian.pape@m-chair.de

Julian Flake
Software Engineering for Critical Systems
TU Dortmund
julian.flake@udo.edu

Andreas Beckmann
Software Engineering for Critical Systems
TU Dortmund
andreas.beckmann@udo.edu

Jan Jürjens
Software Engineering for Critical Systems
TU Dortmund
jan.jurjens@udo.edu

ABSTRACT

We report on an approach and associated tool-support for automatically evaluating and grading exercises in Software Engineering courses, by connecting various third-party tools to the online learning platform Moodle. In the case study presented here, the tool was used in several instances of a lecture course to automatically measure the test coverage criteria wrt. the test cases defined by the students for given Java code. We report on empirical evidence gathered using this case-study (involving more than 250 students), including the results of a survey conducted after the exercises (which yielded positive feedback from the students), as well as a performance evaluation of our tool implementation.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing Tools*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education / Self-assessment*

Keywords

Software Engineering, education and training, case study, automatic grading, software testing.

1. INTRODUCTION

In Software Engineering, it is not only important to teach students software development, but also a structured way for developing test cases. Since it is in general not possible to test all possible combinations of input parameters, it is a challenging task to cover as many relevant parts of the software as possible with a minimal selection of test cases. To measure the degree to which this is achieved, there exist metrics, such as e.g. branch or predicate coverage metrics, or metrics considering the data flow of variables. One of our aims in our advanced Software Engineering course was to teach the students meaning and usage of those metrics. The course consists of 15 weeks with 90-minute-lectures and 45-minute-tutorials. Roughly 200 Computer Science undergraduates participate, typically in their fifth semester.

Several reasons motivated the development of a tool to automatically correct and grade exercises in this context:

- Correcting and grading student's homework on the topic of defining test-cases is a time-consuming, tedious and error-prone task. The resources freed by reducing this burden (at least in the long run) can be used to improve the teaching support for the students in other ways, e.g. by increasing the face-to-face time with the students or by creating additional exercises.
- Automatic assessment overcomes the problem that for different teaching assistants (necessary in the case of higher student numbers as in our lecture), the level of detail of the marking and even sometimes the grading itself could be non-uniform.
- As another advantage of automatic assessments, students may repeat the exercises as often as they like (e.g. for general training purposes or for exam preparation) and receive feedback each time.

An obvious requirement was thus that the tool should ease the teaching assistants' task by correcting the students' homework automatically. Besides that, we identified the following requirements before building the tool:

1. The tool should improve the experience for students by allowing other (more creative) kind of questions beyond multiple choice (which are so far mostly used in a Moodle context) and by giving the students detailed feedback on their solution (not only showing them the number of points they achieved).
2. The tool should allow to specify additional exercises for the students to help them preparing for the final exam without burdening teaching assistants with additional corrections.

ICSE '16 Companion, May 14 - 22, 2016, Austin, TX, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4205-6/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2889160.2889246

3. It should be possible to establish a relationship between accounts in the system and the students' matriculation number in order to give the proper credits in a secure and privacy-compliant way.
4. The source code base which needs to be maintained should be as small as possible (to minimize the long-term effort for this). Thus, the tool should as far as possible rely on existing software engineering tools.
5. The solution should be easily scalable to 400 students, to be able to handle a potential increase in student numbers in the future.

Since the university's computing center already runs an installation of Moodle[1] [1], we based our approach on Moodle. This also had the advantages that students are already accustomed to the system, we would avoid to use an isolated application only in our group, and the administration of Moodle (as well as the user management) would be done by the university's computing center.

In this paper, we present a case study for our online assessment tool. For a first test, we applied our solution to the undergraduate course on Software Engineering in the Winter Semester 2013 / 2014 (WS 13/14). A second application – with large improvements – took place in the Winter Semester 2014 / 2015 (WS 14/15). In both semesters there were six exercises per semester, intended to be performed by the students at home. In WS 13/14 one of these six exercises was carried out by our online assessment tool. In WS 14/15 we replaced two of six paper homeworks by electronic versions and additionally offered the students exercises for exam preparation. The regular exercises were on developing test cases with certain criteria on code coverage (e.g. branch or predicate coverage). The exercises for exam preparation were variations of the previous exercises on testing.

Additionally, but out of the scope of this paper, we implemented automated assessment of exercises regarding the use of the Object Constraint Language (OCL) in the context of models in the Unified Modeling Language (UML).

In WS 14/15 we did an extensive survey to get feedback from the students on this automated assessment system. Most feedback was positive, in accordance with work by Fox [2] and Burge et al. [3], stating that using blended learning technology (such as offereing recorded lectures and using automated assessment) in a conventional on-campus lecture can significantly improve the quality of a lecture.

The paper is structured as follows: In Sec. 2 we discuss related work. The system architecture is explained in Sec. 3 and the exercises given to the students in Sec. 4. In Sec. 5 the evaluation of our system is presented and Sec. 6 discusses future work and concludes our work.

2. RELATED WORK

There is a lot of work on automatic assessment of programming exercises dating back to 1960 [4] and 1965 [5]. In general, those tools receive the students' source code and/or binary as input and automatically assess and rate the correctness and quality of the submitted code. For an overview about those tools, we refer to the surveys [6, 7, 8].

Evaluation strategies based on test cases with criteria for code coverage seem to be very close to our work. However, even though sometimes the same tools are used that we use in our tool-set (such as CodeCover), our aim is different. The focus of those tools is to assess the quality of the students' code and give them feedback how their code performs with the aim to develop the students' programming skills. Instead of expecting source code from the students, our tool presents a piece of code to the students and expects them to enter some test cases fulfilling different criteria for code coverage. Thus, the aim of our tool is to teach the students to get an understanding of the different criteria of code coverage. Although Edwards and Shams argue that for the evaluation of the quality of tests more sophisticated techniques should be put in place [9], in our opinion teaching the students different criteria of code coverage and their limitations provides the ground for further education.

The for-profit educational organization Udacity offers a course on software testing[2], including automatically assessed testing exercises. However, their exercises are integrated within their own MOOC-platform and there seems to be no publically available information about its implementation.

Regarding the integration in Moodle there exists work which connects external programs to the Moodle learning platform. Barana et al. report on a connection of the mathematic analysis software Maple to the Moodle platform using a plug-in for automatic assessment of exercises [10]. There are already several use cases of Moodle plug-ins in Software Engineering courses. As shown by Sun Zhigang et al. [11] there is a variety of Moodle plug-ins which, when combined, can significantly improve the students learning experience and decrease the workload of an instructor. For example, Sanchez et. al. present an extension to Moodle for automatically assessing database exercises [12]. However, the majority of plug-ins aim to automatically grade student submitted code (such as CodeHandIn [13], or CodeRunner [14]).

Not only plug-ins have been used to extend Moodle. The development of OPAQUE (cf. Sect. 3.2) and its integration into Moodle goes back to OpenMark[3], a computer-assisted assessment system. Therefore, Moodle can also be connected with OpenMark. There is also another Question Engine (cf. Sect. 3.3) ounit[4], a unit testing environment that can be used to evaluate programming assignments. However, it seems not to be actively maintained at the moment.

3. SYSTEM ARCHITECTURE

As already discussed in Sect. 1, the most principal design decision was dominated by the organizational setting. At TU Dortmund there is a central installation of the web-based e-learning platform *Moodle*. Any enrolled student is able to use this Moodle instance by using a single-sign-on system along with his or her personal university account. Moodle has sophisticated capabilities regarding basic requirements of an e-learning platform. Beside others, the capabilities we are interested in are user authentication, test composition, test execution and reporting functionalities. To avoid extra guidance and maintenance efforts by introducing an additional system, such as instructing students to use another system or platform for one specific lecture, we decided to make use of the university's central Moodle installation. An additional advantage is that the Moodle installation is maintained by the university's computer center, including

[1]https://moodle.org

[2]https://www.udacity.com/course/software-testing--cs258
[3]https://java.net/projects/openmark
[4]https://github.com/anttix/ounit

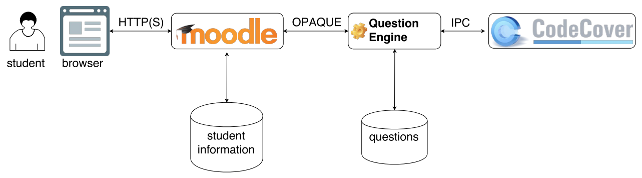


Figure 1: Architecture overview

the identity management. To make use of these advantages, we chose Moodle rather than another e-learning platform.

Wrt. the requirements from Sec. 1, Req. 3 (user identification) is supported by the central installation and its single-sign-on system. Considering Req. 1 (sophisticated question types), the question types directly supported by Moodle do not satisfy our demands. As a consequence, an external tool was needed. Given Req. 4 (minimal code base to be maintained), we chose the (at that time actively developed) third party tool CodeCover. Req. 2 is fulfilled by Moodle's capability of organizing exercises. Approaches to fulfill Req. 5 are discussed in Sec. 3.5 and its evaluation in Sec. 5.2.

The overall architecture is shown in Fig. 1. The students access the system with their browser, Moodle uses a protocol named *OPAQUE* to connect third-party software (in our case CodeCover) to evaluate students' answers. Since CodeCover has no interface to the OPAQUE protocol, a direct connection was not possible and we needed to develop the Question Engine as a middle layer.

In the place of the component CodeCover in Fig. 1, one can of course also connect other software. For example, we have already connected an OCL checker, to support automated assessment of exercises regarding the use of the Object Constraint Language (OCL) in the context of models in the Unified Modeling Language (UML). Due to space restrictions, this is beyond the scope of the current paper.

The remainder of this section is organized as follows: In Sec. 3.1, we give further details on Moodle. In Sec. 3.2 and Sec. 3.3, we describe the OPAQUE-protocol and the associated Question Engine. We then briefly discuss how CodeCover was addressed by the Question Engine in Sec. 3.4. Our load distribution setup is sketched in Sec. 3.5.

3.1 Moodle

Moodle is an online learning management system, written in PHP and published under the GNU Public License (GPL). The Moodle server is responsible for the authentication of users, managing enrollments, organizing test availability, ordering of contents inside the tests, and recording marks.

As explained above, the question types directly supported by Moodle did not satisfy our demands. Moodle is extensible by plug-ins, and there are a lot of plug-ins available in Moodle's central plug-in repository that implement additional question types. However, since any additional plug-in requires additional maintenance and may be a potential threat to the stability and security of the Moodle instance, the installation of such plug-ins was not possible according to policy of the university's computer center. Also, because most of them would be specific for a single lecture and not useful for the majority of groups using the central Moodle installation, this would additionally lead to an enhanced complexity for all lecturers when configuring the exercises, due to an increased number of question types. Moreover, plug-ins causing a high load or memory consumption, and

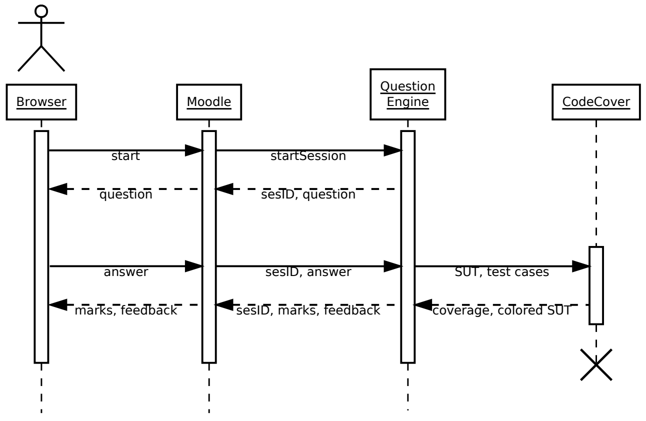


Figure 2: Sequence of messages between browser, Moodle, Question Engine and CodeCover

thus affecting the performance of the overall system are undesirable on the central server.

It was, however, possible to install two plug-ins enabling Moodle to communicate with Question Engines. The plug-ins act as a client of a SOAP-based web-service provided by one or more Question Engines. Those plug-ins do not include functionality specific to a particular lecture. The functional aspect of the plug-ins is very limited and generic, so only little or no modifications are expected in the future. The plug-ins are available through the central Moodle plug-in repository, they are updated within the university's computer center (Moodle-)update process, which takes place at a regular basis. Therefore, only little extra maintenance effort is expected. Moodle only acts as a client and does not need to offer additional services over the network.

Concluding these considerations, the solution described above allowed us to make use of the centrally maintained Moodle installation while we were not limited to Moodle's built-in question types. Responsibilities are strictly separated: the Software Engineering group is responsible for the Question Engine, while the central computing center maintains the central Moodle installation (including security- and privacy-sensitive data, such as user accounts). Since the Question Engine operates on its own server, availability and performance of the central Moodle instance are not affected by potential issues due to computationally intensive tasks which may arise from the sophisticated automated assessment using third-party components.

3.2 The Open Protocol for Accessing Question Engines

The Open Protocol for Accessing Question Engines (*OPAQUE*[5]) is a SOAP-based protocol used by a *test management system (TMS)* (in our case Moodle) to communicate with a *Question Engine (QE)* which provides content and performs or delegates assessments of students' answers. The Question Engine is described in more detail in the next section. OPAQUE is used when questions are fetched from the Question Engine and the students' solutions of the exercises are assessed. We only give a brief description and refer to the website mentioned above for more details and a full description of the Application Program Interface (API) and

[5]https://docs.moodle.org/dev/Open_protocol_for_accessing_question_engines

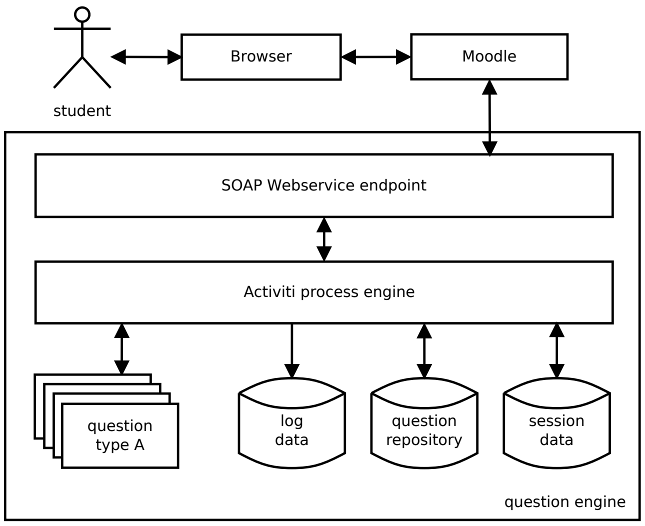


Figure 3: Components of the Question Engine

the OPAQUE Web Services Description Language (WSDL, an XML format for describing network services).

Fig. 2 shows a possible sequence of messages. After the session is started by Moodle, the Question Engine generates a session ID (sesID). Moodle starts a new session for every student and every question the student works on. All subsequent messages between Moodle and the Question Engine for a given student and the concerned question contain this session ID. This results in a stateful protocol, which allows the Question Engine to maintain a complex state. The diagram also shows that the question is created by the Question Engine. After the student has worked on the exercise in his browser, the answer is then assessed by the Question Engine. The Question Engine may use additional software or services to do (or assist in) the assessment. The feedback and the mark is then sent back to Moodle.

3.3 Question Engine

The Question Engine handles the rendering of questions, the grading of responses and the generation of feedback to the students' solution of the exercises. Our implementation of the Question Engine is based on the Activiti BPMN2.0 Process Engine[6]. Its components are shown in Fig. 3.

As described in Sect. 3.2, OPAQUE is a stateful protocol. To assure reproducibility, the Question Engine is required to guarantee that a sequence of calls with exactly the same arguments will always return the same results. This way, one can always recreate a question session simply by playing back the sequence of web service calls. This may be needed in case the students interrupt working on the exercises or in case the Question Engine crashes. Therefore, the state of the question session is stored by the Question Engine. By using the Activiti Process Engine, one process instance is created per question session and all relevant variables of a question session are persisted automatically by the process engine. Another useful feature of utilizing the Activiti process engine is that every event may be logged to a database, so users' behavior and resulting load distributions over time can be analyzed afterwards. The analysis results of the performance evaluation of the Question Engine are described in Sec. 5.2.

Our implementation of the Question Engine offers two functionalities. For some of the questions, the Question Engine handles the creation of question as well as the grading by itself (cf. Sec. 4). For other questions, the Question Engine acts as a front end to an external tool. In the latter case, the Question Engine's main task is the transformation of Moodle's request to a proper call to the external tool. Conversely, the tool's evaluation is translated back before being sent to Moodle. Since the external tools are usually not used for grading, the Question Engine also has to grade the student's answer based on the evaluation of the tool.

In this paper, we focus on automated assessment of the definition of test cases and therefore describe the connection to CodeCover in the next section. In general, this set-up can be applied to other external tools and we successfully also connected our implementation to the Eclipse OCL checker[7] to assess the correct use of the Object Constraint Language (OCL) in the context of models in the Unified Modeling Language (UML).

3.4 CodeCover

CodeCover[8] is a tool that measures several code coverage metrics in the context of white-box testing. CodeCover has been released under the Eclipse Public License (EPL), its Java source code is available.

The choice of CodeCover was the result of an evaluation of tools measuring test coverage, using the following criteria:

1. Support of several different coverage metrics: While a lot of software in the field of white-box testing determine statement coverage and branch coverage, only few tools cover metrics beyond these two. Since we are interested in teaching a broad variety of metrics, we paid particular attention to this criterion.
2. Possibility to programmatically process measurement results in order to derive marks from the degree of coverage: CodeCover implements a function to create a comma separated value (CSV) file that contains the metric values at a detailed level.
3. The goal to provide valuable feedback (Req. 1 from Sect. 1): CodeCover is equipped with a function to create an HTML snippet of the measured source code where covered parts of the source code are distinguished from uncovered parts of the source code by highlighting the source code in different colors. This function was a great opportunity to support students' understanding of the coverage metrics by providing individual feedback.
4. Availability of the software's source code (in case we later intend to add additional coverage metrics): CodeCover's source code is available under the EPL.
5. Active maintainance: We did not consider software that has not been actively maintained for a while. At the given time, CodeCover was under active development and maintainance.

CodeCover can be run in two different ways, inside the well known Eclipse IDE or inside an Ant build. We did not follow the Eclipse plug-in approach, since the dependencies include GUI widgets and interactions which are not realizable in a headless environment. We made use of the second approach, the possibility to invoke CodeCover inside an Ant

[6]http://activiti.org/

[7]http://www.eclipse.org/modeling/mdt/?project=ocl

[8]http://codecover.org

build. Ant is a build tool for Java software. It expects a project description that configures the different parameters and steps of the build process. A step in a build process is called *task* (e.g. *run the Java compiler* or *create JavaDoc*). Different tasks are aggregated into a sequence, called *target.* A minimal Ant project description describes one target that contains one task. CodeCover implements an Ant task definition, so Ant targets can be configured to include the execution of CodeCover. Normally, Ant builds are triggered from command line or by an IDE, but there also exists the possibility to invoke Ant programmatically within other programs, which is the method we chose.

In order to use CodeCover as an assessment tool inside our environment, several tasks like transformation, extraction or enrichment of information have to be performed by the Question Engine. CodeCover evaluates a JUnit test suite against an instrumented version of the *software under test (SUT)* and produces reports containing the evaluation results. We therefore need to provide an instrumented version of the SUT and JUnit test cases as inputs, and perform post-processing of the reports given as output.

Instrumenting source code means inserting additional statements at certain places into the source code (e.g. after every statement). It is sufficient to do this once per question (each question may refer to different SUTs). Instrumentation of the SUT is triggered by the Question Engine during its initialization phase and performed by CodeCover. The students' responses describe the test cases using a simple mathematical notation formalized by a simple grammar. The grammar is described by an ANTLR[9] grammar and the responses are parsed by an ANTLR generated parser.

From the parsed information (a set of parameter tuples), method calls are generated and wrapped into a JUnit test suite, which has to be compiled afterwards. While the compiled test suite is run by JUnit against the instrumented version of the SUT, the data required by CodeCover is gathered. After running the test suite, CodeCover is called to create reports using the gathered execution data. After CodeCover has created the reports (CSV and HTML), the Question Engine picks them up to calculate marks from the values in the CSV report and to create an individual feedback to the student containing colored source code.

3.5 Load distribution

Since an evaluation and assessment of students' responses using sophisticated third-party software could be expected to become more complex and resource demanding than comparing strings or checking multiple choice questions (the usual assessment in a Moodle context), a scalable solution was important. Fortunately, the OPAQUE client plug-ins for Moodle offer the possibility to realize a simple load distribution mechanism. During configuration of a Question Engine, multiple endpoints can be configured. When the Moodle plug-ins start to establish a connection to the Question Engine responsible for a certain question, one endpoint from the set of endpoints is selected randomly. Using this mechanism, we can use multiple installations of the Question Engine, all providing the same service and questions.

For WS 14/15, we expected an increased number of participants in the course compared to WS 13/14 and finally ran eleven Question Engine instances on virtual hardware in a private cloud environment to distribute load. Moodle was configured accordingly to query one of these eleven instances by random selection. We installed a MySQL[10] server on an additional virtual machine that acted as a common data store for the eleven Question Engine instances.

4. EXERCISES

For our courses in WS 13/14 and WS 14/15, we had several questions related to different paradigms and approaches in the software testing field. Some of these questions made use of Moodle's built-in question types, others were delivered by the Question Engine we developed.

Although Moodle offers multiple choice questions, simple calculation (division and rounding) and comparison of numbers, there are some restrictions of Moodle's built-in question types (e.g. to realize "all or nothing" multiple choice questions). Therefore, we used some question types implemented by the Question Engine in order to circumvent those restrictions and to gain more flexibility regarding the presentation. For example, in one of the exercises we presented source code and a test case and asked which instructions of the source code are executed or which degree of code coverage is achieved by the test data.

Other questions delivered by the Question Engine made use of more sophisticated assessment mechanisms enabled by CodeCover. Furthermore, the usage of CodeCover allowed us to give more detailed feedback to the students. Therefore, in this section we present one particular question type which makes use of CodeCover as an assessment tool

The goal of this question type is to assess the students' knowledge of how the different code coverage metrics in the context of white-box testing differ from each other and how they are calculated. In the end they should be able to select a minimal set of test cases for a given *software under test (SUT)* to reach a high code coverage (while being taught that a high code-coverage by test-cases in itself does not necessarily lead to good software, of course).

4.1 Example on Full Statement Coverage

For illustration, we describe a simple example that asks the students to give a *minimal set of test cases* that reaches a *full statement coverage* of a given SUT source code.

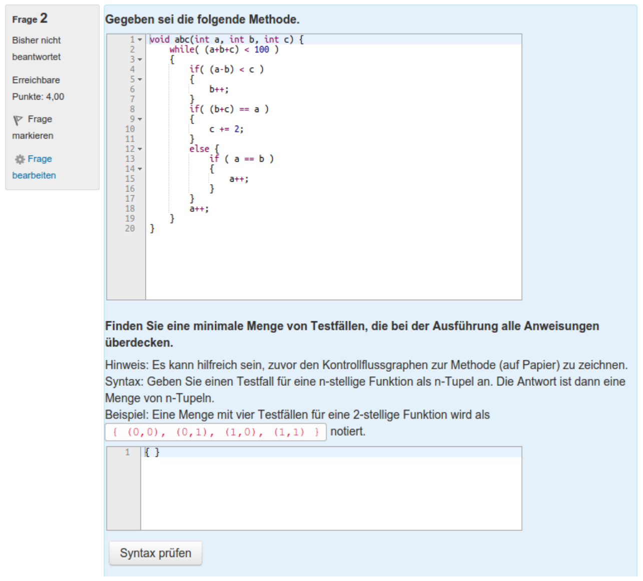
Fig. 4 contains the question as it is shown in an exercise inside Moodle. At the top, the window displays the source code of the SUT. The source code is followed by a German description of the students' task and instructions what they should do. A rough translation of the instructions reads as follows: "Provide a minimal set of test cases that reaches a statement coverage of 100% of the source code."

The students are instructed to type their answer into the editor field at the bottom, by using a simple formal syntax described in the instructions. Students can check their answers for syntactical correctness before submitting it by pressing the button "syntax check" below this editor field. That way we reduce the number of non-processable answers. A (syntactically correct, but semantically incorrect) answer to our example is: $\{(1,2,0),(2,1,1)\}$, which (if correct) would mean that the set consisting of the two test cases $(1,2,0)$ and $(2,1,1)$ indeed reaches a statement coverage of 100% of the SUT and that there is no smaller set of test cases which would reach a full statement coverage of the SUT.

As described in Sec. 3, Moodle receives this answer and

[9]http://www.antlr.org/

[10]http://www.mysql.com/



Gegeben sei die folgende Methode.

```
void abc(int a, int b, int c) {
    while( (a+b+c) < 100 )
    {
        if( (a-b) < c )
        {
            b++;
        }
        if( (b+c) == a )
        {
            c += 2;
        }
        else {
            if ( a == b )
            {
                a++;
            }
        }
        a++;
    }
}
```

Finden Sie eine minimale Menge von Testfällen, die bei der Ausführung alle Anweisungen überdecken.

Hinweis: Es kann hilfreich sein, zuvor den Kontrollflussgraphen zur Methode (auf Papier) zu zeichnen.
Syntax: Geben Sie einen Testfall für eine n-stellige Funktion als n-Tupel an. Die Antwort ist dann eine Menge von n-Tupeln.
Beispiel: Eine Menge mit vier Testfällen für eine 2-stellige Funktion wird als `{ (0,0), (0,1), (1,0), (1,1) }` notiert.

```
{ }
```

Figure 4: Example of a question using CodeCover

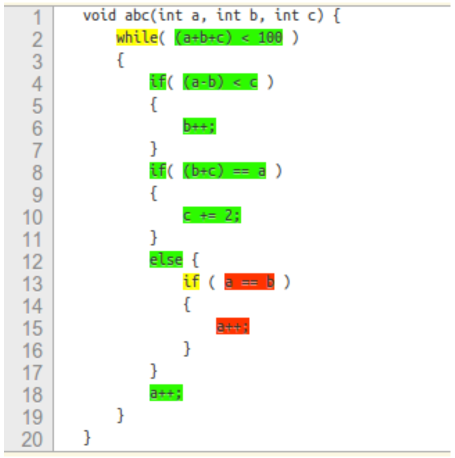
hands it over to the Question Engine. The Question Engine parses and syntactically modifies the answer and hands it over to CodeCover. CodeCover determines the requested metrics and illustrates the covered and uncovered parts of the SUT's source code by using different colors. The Question Engine takes CodeCover's results, calculates the gained marks according to configured weights for different criteria and sends the results back to Moodle. Depending on the configuration, there are two possibilities:

Direct feedback: The students can immediately see their marks and the colored source code (Fig. 5) as part of an individual feedback (useful in particular when using the tool independently from the official tutoring).

Deferred feedback: The evaluation is only available after a given deadline, in order to discourage students from copying the answers from other students.

5. EVALUATION

To evaluate to which degree the implementation fulfills the requirements defined in Sec. 1, we performed a survey with the students and measurements of the system's performance.



```
void abc(int a, int b, int c) {
    while( (a+b+c) < 100 )
    {
        if( (a-b) < c )
        {
            b++;
        }
        if( (b+c) == a )
        {
            c += 2;
        }
        else {
            if ( a == b )
            {
                a++;
            }
        }
        a++;
    }
}
```

Figure 5: Colored source code (covered and uncovered parts)

5.1 Survey

To measure the students' acceptance of online exercises and especially our implementation, we conducted a survey.

5.1.1 *Participants and General Setup*

Our tool was used in an undergraduate course on software design, usually taken by computer science students in the fifth semester of their bachelor studies.

Initially, we used it in the Winter Semester 2013/14 (WS 13/14). A second application took place in the Winter Semester 2014/15 (WS 14/15). In both semesters, there were six exercises per semester, intended to be performed by the students at home. In WS 13/14, one of these six exercises was carried out using our online assessment tool. In WS 14/15, we replaced two of six paper homeworks by electronic versions using the tool.

The students had to achieve at least 50 percent of the overall score of the exercises, and 30 percent of of the score for each subsequent pair of exercises (1/2, 3/4, and 5/6) in order to be accepted to the final exam for the lecture course. The latter condition was introduced to ensure that students will work on some exercises throughout the whole semester.

Table 1 shows the number of students participating in the courses' regular exercises. Online exercise participation numbers are printed in bold. Only those participations in online exercises are taken into account where the exercises have been completed and submitted in time. Only one submission per person has been counted.

The decreased participation in the sixth exercise in both terms is most likely because many students already gained the necessary scores to participate in the course's final exam.

In WS 14/15, we additionally provided four exercises which students could perform voluntarily in order to prepare for the final exam. The additional exercises 1 and 2 were slight variations of the regular exercises 5 and 6 in WS 14/15 (same type, but different instances). The additional exercises 3 and 4 provided contents that were covered by paper versions in the regular exercises during the semester (e.g. OCL). The number of participations in these optional, automatically assessed exercises are also shown in table 1.

5.1.2 *Questionnaires*

The questionnaires were designed to answer the following questions: Were the online exercises more or less demanding than the traditional exercises? How were the exercises perceived by the students? Would the students prefer more or less online exercises for future lectures? Were there any technical obstacles while working on the exercises? Were the additional voluntary exercises a helpful addition regarding the preparation for the final exam?

To answer these questions, the questionnaires were split into three sets of questions. The first set of questions was aimed at gathering basic information about the students. What is their major subject, how long are they studying and did they already visit the course in a previous year?

The second set of questions targeted the amount of time the students spend working on the lecture and the percep-

Table 1: Number of students participating in exercises

Exercise	1	2	3	4	5	6
Regular, WS 13/14	127	122	101	**99**	118	52
Regular, WS 14/15	121	147	144	146	**148**	**81**
Addit., WS 14/15	**64**	**83**	**113**	**108**	–	–

tion of the online exercise. The students could answer these questions using the Likert-type scale system [15], thus generating quantitative examinable data.

The last set of questions was used to ask the students which parts of the online exercises were particularly good or needed improvement. The students could answer these questions using free text fields generating qualitative examinable data. At the end of the survey, the students were asked to give a grade representing their overall opinion of the online assessment system.

We used two different questionnaires: One was presented after the students submitted their solutions, while the second was presented after the students viewed the evaluation and score of their answer. The questions differed slightly depending on these contexts: The first questionnaire focused on the submission process and assignment, and the second on the feedback given by the system.

5.1.3 Procedure

Students were asked to participate in the survey after submitting their solutions to the Question Engine and after viewing their scores. Along with the request for feedback, we presented the students a link to a Limesurvey server, hosted by the university. Limesurvey[11] is a tool to develop, publish and collect responses to surveys, written in PHP and published under the GNU Public License (GPL).

The participation was voluntary and anonymous. No measures were taken to avoid multiple survey submissions by the same student because this was not practically feasible without compromising anonymity. Also, no connection between exercise scores and survey was made, which also would have compromised the anonymity of the students.

It was possible to take part in the surveys regarding the exercises as long as the exercises were open to be solved. The surveys regarding the feedback of the Question Engine were open from the moment the feedback was made public until the date of the first final test. This date was five weeks after the feedback for exercise 5 was made public and three weeks after the feedback for exercise 6 was made public.

Altogether, we received a total of 105 completed questionnaires. We only considered completed questionnaires, but it was possible to skip some of the questions.

5.1.4 Quantitative Analysis

We asked the participating students about their demographics, and as expected, most answers reported them to be in the 5th to 7th semester (73%), while 13% were repeating the lecture. All of them studied one of the two main courses in the department (Computer Science and Applied Computer Science). 40% the answers stated that the students regularly visited the lecture (consistent with the fact that video recordings of the lecture were made available along the semester and there is no mandatory attendance at the lectures), while for the on-site tutorials it were 80%.

Table 2 shows an excerpt of the questions with a Likert-type scale. For some of the questions, we also asked their negation to ensure that the students were not just selecting the same answer for all questions. Some of the questions (e.g. regarding exam preparation) were only part of the last questionnaire and the students were able to skip single questions. Thus, not all questions share the same number of answers.

[11]https://www.limesurvey.org/

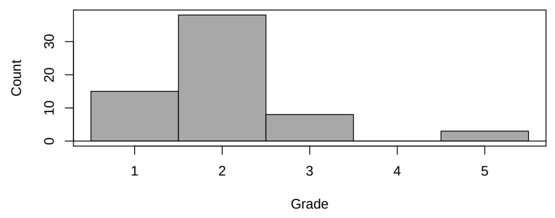
Table 2: Answers to questions with Likert-type scale

Question	++	+	o	-	- -
Online exercises required more effort than paper exercises.	6	3	18	23	15
The motivation to work with online exercises was higher than with paper exercises.	11	19	14	7	12
When working on the exercises, technical problems occurred.	3	2	4	5	53
The usability of the online system was good.	43	25	4	1	3
The feedback was helpful for understanding the exercise.	11	8	9	4	2
Feedback for online exercises was more detailed than for paper exercises.	3	3	6	5	6
Online exercises were helpful for exam preparation.	7	4	0	0	0
Overall, I preferred the online exercises.	37	27	17	8	10

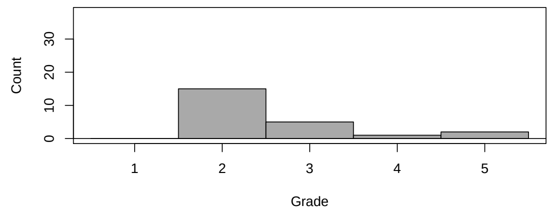
In summary, technical problems or usability of the system were no issue. Feedback seemed to be helpful for most students, but it could be more detailed. The motivation to work on online exercises seems to be roughly the same than for paper exercises. However, the majority of answers preferred the online exercises, perhaps because they require not more (but maybe less) effort than paper exercises.

At the end of the survey, we asked the students to rate our online exercises with a school grade, where 1 (very good) reflects the best mark and 5 (poor) reflects the worst mark. We distinguish between grades submitted after working on the exercises (Fig. 6a) and grades after viewing the feedback and the score of their formerly transmitted answer (Fig. 6b). This shows that the students were more confident with the system after the exercises. The average of the students rating of our system was 2.1. This indicates that the students response in general was positive.

5.1.5 Qualitative Analysis



(a) Grades by the Students after Working on the Exercise



(b) Grades by the Students after Viewing the Feedback

Figure 6: Students' Grades of our System

The goal of the qualitative analysis was to evaluate the free text answers of the students to extract qualitative assertions. Those assertions add to the quantitative analysis and allow a more detailed look at the students mindset towards the system.

In this analysis, we examine all responses to the different questionnaires as a single data set. This procedure had no negative impact on the accuracy of the findings of the analysis but allows a more concise presentation.

Following best practices as described by Miles, Huberman and Saldaña [16], we first examined each answer for any potential feedback regarding the technical implementation of the system. Using the rules described in Table 3, the answers were individually categorized as either positive, neutral or negative. This procedure was repeated for feedback with regards to the content of the assignments.

The accumulated occurrence of each category is shown in Table 4, where the positive, neutral, negative or missing feedback regarding the content of the exercises (row headers) is set in relation to the positive, neutral, negative or missing feedback regarding the technical aspects of the system (column headers). Even though most qualitative feedback about the content of the exercises was negative (56%), the technical feedback is mostly positive (39%) or neutral (43%). For example, 6 questionnaires contained negative feedback on the content but positive feedback on the technical aspects, and 0 questionnaires the other way around.

In a second evaluation step, each answer was analyzed for reoccurring topics. The most common topics were (with the accumulated occurrences in brackets):

+ Advantages of digital submission (18): This topic combines statements about the advantages of being able to submit solutions digitally compared to submitting on paper. This includes advantages like the ability to submit multiple solutions over time, working completely paperless or saving a commute to the university.
+ Precise feedback (5): The feedback given by the software after grading was perceived as more precise than feedback given in traditional corrections.
+ Intuitive usability (5): The usability of the software was described as easy to use and intuitive.
- Editing of submitted answers (7): Critique towards the user interface, for example the missing option to change parts of an answer during a resubmission.
- Similarity of Assignments (6): Students complained about a lack of difference between the assignments and expressed the wish to get a bigger variety of different assignments.

Table 3: Rules used to determine the general categories

Category	Description
Positive	The feedback of the answer was mainly positive. If there was any negative feedback in the answer, it was significantly outweighed by the positive feedback.
Neutral	The feedback was fairly even in being a positive or negative critique. No real tendency was recognizable.
Negative	The feedback was mainly negative. If there was any positive feedback in the answer, it was significantly outweighed by the negative feedback.

Table 4: Result of categorization of Feedback

Content\Technical	Pos.	Neutr.	Neg.	No Feedback
Positive	4	1	0	1
Neutral	3	5	0	1
Negative	6	6	2	5
No Feedback	6	9	7	48

- Bad Performance (5): This topic consolidates statements regarding a negative experience caused by bad performance of the back end. This issue is further discussed in Sec. 5.2.
- Indifferent Grading (5): The fear of students to miss out on points that a human instructor would possibly allocate but the software would not. This could be the case if, for example, the student showed a general understanding of the topic but made a mistake in the progress.
- Viewing the exercises (4): The wish to be able to view the exercises without being connected to the system, e.g. as a downloadable file.

The topics regarding positive effects of the system towards the students (marked with a plus sign as a bullet point in the above list) are more prominent (55%) than critical topics (marked with a minus), which can be interpreted as a positive opinion of the students towards the system. As a first conclusion, the students seem to have well adopted to the new system and have a positive mindset towards a broader integration of online exercises into the curriculum. On the other hand, there also seems to be room for improvement. We further discuss this in Sect. 6.

5.1.6 Threats to Validity

As mentioned, it was important to us to ensure the participants' anonymity when participating in the surveys. We also did not want to make the participation in the survey mandatory. Therefore, we only know about the feedback of the students who voluntarily participated in at least one of the surveys. We also did not want to risk that the students could be afraid that we are able to revoke their anonymity and therefore we did not try to ensure that a student did not take part multiple times or try to link the feedback of a specific student between the different surveys. Both decisions could have numerous effects on the collected data sets, as it would be possible for students to do the survey multiple times. One student only gave positive feedback, but graded the system with poor. Thus, we cannot rule out that students misunderstood the way they should score the system, leading to wrong marks.

All qualitative data analysis of free text responses is prone to a biased perspective of the analyzing person and a misunderstanding of the original meaning of a response. For example, this could result in a disregard for negative comments towards the system as a person might falsely blame the users incompetence for not understanding the user interface of the system. To avoid any makers bias, the qualitative analysis was primarily carried out by a person not affiliated with the development of our Question Engine. To counteract any residual bias possibility of a single person, the categorizations were collectively discussed and reviewed.

To validate the categorization, we used the students rating of the system to cross reference whether a negative comment was more likely to have been made by a student who rated the Question Engine negatively. This was true, which further indicates that the categorization used here is valid.

Due to the fact that the mean rating of the students who did *not* give any feedback was about 1.1 (i.e. rather positive), the qualitative analysis will be skewed towards more negative comments, which has to be kept in mind wrt. the relevant discussion in Sec. 5.1.5.

Due to the requirement to gain a certain amount of points during the semester to be able to take part in the final test after the semester and the fact that exercise 6 was the last exercise in the semester, most students already had the required scores and did not take part in the last exercise. This resulted in a significantly lower survey count for exercise 6 and the subsequent survey which was shown after students reviewed their scores. Due to the fact that many of these students still needed to gain a certain amount of points, an emotional reaction to seeing the received score is possible. This could either have lead to a more positive view towards the Question Engine (e.g. the grading was surprisingly good) or to a more negative view (e.g. if the student's result was less then what was required to take the final test).

5.2 Performance

To set the context to discuss our system's performance, we regard the users' behavior. As expected, the submissions of the students' answers were not evenly distributed neither over day-time nor over the editing time for the assignment.

Figure 7a shows the distribution of the requests (denoted as service tasks) per hour of day. The number of requests per hour increases until noon and then decreases until the end of the day. To discourage submitting closely before the deadline (and avoid a corresponding server overload), we fixed the deadlines to 6 am.

Figure 7b shows the distribution of requests over the whole period. Four peaks can be identified. The two peaks in mid of January and beginning of February mark the deadlines of the assignments. The two peaks in end of February and end of March mark the days before final exams.

The number of submissions increased significantly before the deadlines. For example, in WS 13/14 about two-third of submissions took place in the last 48 hours, while the students had 17 days in total to complete the exercise. The reason is not that the exercises were too time-consuming, because about two-thirds of the students submitted their answers the same day they started their exercises.

For the performance analysis it is necessary to have a closer look at the requests to the Question Engine. Each homework consisted of several questions. Each answer to a question produced at least two requests that had to be handled by the system. We allowed the students to check their answers for syntactical correctness. For this check, the students had to type their answers and click on a *syntax check* button. This button triggered the system to evaluate the answer for processability and return the evaluation results without disclosing whether the answer was correct in regard to the question. To provide this function, a synchronous messaging had to be established and the responses of the system had to happen within an acceptable time frame (we defined 20 seconds as an acceptable time frame). Meeting this constraint became challenging for those questions where a syntax check triggered some computational effort (like parsing and possibly fully evaluating responses for processability).

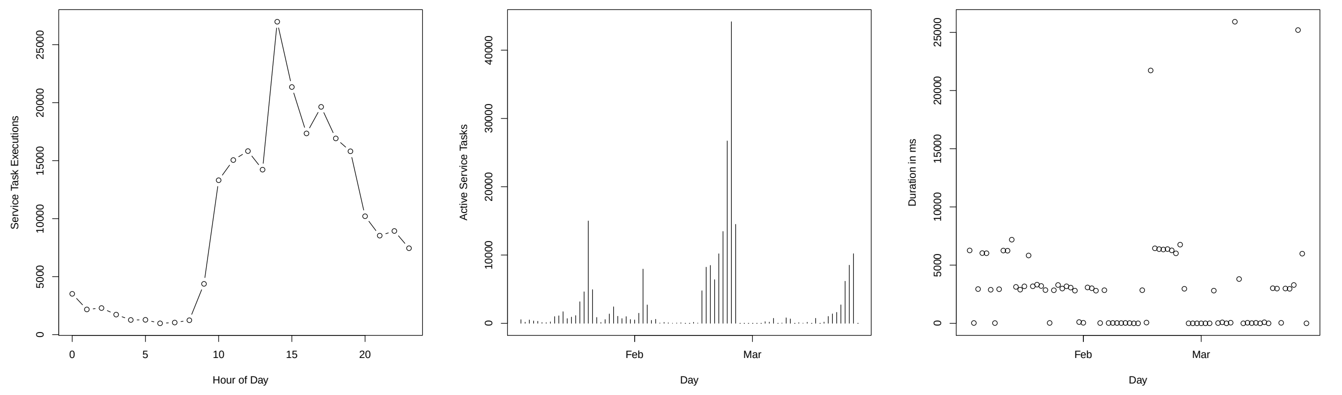
Figure 7c shows the maximum durations per day taken to process a student's response. The three days with maximum service task durations above 20000 milliseconds stem from the data-log hard-drive running out of space. We underestimated the amount of log data for the analyses described here. We put countermeasures in place to avoid these incidents in future and do not further discuss these outliers here. For the remaining observations in Fig. 7c, one can observe three different levels of maximum durations per day. The first level of duration is made of points near 0, that means that on these days, no request lead to an expensive processing of a response and all request were processed within less than 150 milliseconds. The second level is at about 3000 milliseconds. On these days, requests were processed where some source code compilation took place. The third level is made of days with a maximum service task duration of about 6000 milliseconds. These are days when two computationally intensive tasks were performed at the same time on the same virtual machine. If there were three computationally intensive tasks to be performed at the same time on the same virtual machine, we would expect a maximum service task duration of about 9000 milliseconds, but this never happened. Ignoring the three outliers, we conclude, that students did not have to wait for more than seven seconds to get the result of a request during the test. Therefore, we consider our system's performance as sufficient and, due to the load balancing discussed in Sect. 3.5, we expect it to be scalable to at least twice the number of students.

6. CONCLUSION AND FUTURE WORK

We reported on an approach and tool-support for automatically evaluating and grading exercises in Software Engineering courses based on Moodle. In the case study presented here, the tool was used in several instances of a lecture course to automatically measure the test coverage criteria wrt. the test cases defined by the students for a given Java code. Additionally, but out of the scope of this paper, we implemented automated assessment of exercises regarding the use of the Object Constraint Language (OCL) in the context of models in the Unified Modeling Language (UML).

Empirical evidence involving more than 250 students suggests that most students had a positive or neutral view towards the Question Engine. Even the neutral responses can be considered good, since the automation of the assessment still allow us to free up teaching resources to improve the teaching in other ways.

Based on the feedback, we plan to further enhance the system. Besides minor improvements, such as further improving the performance of the assessment of the students' answers or facilitating the editing of answers, we are working on the following improvements: We plan to create additional exercises for testing, e.g. employing more code coverage metrics. Additionally, we will further improve the feedback, e.g. by automatically creating control flow graphs with colored branches for more individual feedback. Creating different instances of the exercises for each student will allow us to give instant feedback on the students' answers without the fear that students could copy their homework from their fellow students. Furthermore, it would improve supply with additional exercises for the students' exam preparation. We also plan new exercises by connect other software engineering tools via the Question Engine to Moodle, such



(a) Service Tasks by Hour of Day (b) Number of Service Tasks per Day (c) Max. Service Task Durations per Day

Figure 7: Different Metrics Measuring the Question Engine's Load

as CARiSMA[12], a compliance, risk, and security analysis tool of software models, developed at our group, which may already be used as a web service via SOAP.

There are efforts to adapt Moodle to mobile devices [17]. We plan to adapt our exercises accordingly. As a result, it would be possible to use our system as an audience response system during lectures. Currently, most of the audience response systems suffer the same limitations for software engineering specific exercises as previously described for Moodle. Therefore, we plan to use the system for direct feedback on currently taught topics in the lectures. Using short exercises with a quick overview of the students' results, the lecturer may adapt his lecture to the audience.

Summing up, we consider using online exercises and our implementation successful and plan to further advance it.

7. ACKNOWLEDGEMENTS

This work was partially funded by the projects Visual Privacy Management in User Centric Open Environments (EU, proposal no. 653642) and SecVolution@Run-Time, DFG Priority Program 1593 (DFG project no. JU 2734/2-2).

8. REFERENCES

[1] A. P. F. F. Lopes, "Teaching with Moodle in higher education," tech. rep., Institute of Accounting and Administration (ISCAP), Polytechnic Institute of Oporto (IPP), 2011.

[2] A. Fox, "From MOOCs to SPOCs," *Communications of the ACM*, vol. 56, no. 12, pp. 38–40, 2013.

[3] J. Burge, A. Fox, D. Grossman, G. Roth, and J. Warren, "SPOCs: what, why, and how," in *46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pp. 595–596, ACM, 2015.

[4] J. Hollingsworth, "Automatic graders for programming classes," *Commun. ACM*, vol. 3, pp. 528–529, Oct. 1960.

[5] G. E. Forsythe and N. Wirth, "Automatic grading programs," *Commun. ACM*, vol. 8, no. 5, pp. 275–278, 1965.

[6] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming: A review," *J. Educ. Resour. Comput.*, vol. 5, Sept. 2005.

[7] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, (New York, NY, USA), pp. 86–93, ACM, 2010.

[8] J. C. Caiza and J. M. D. Alamo, "Programming assignments automatic grading: Review of tools and implementations," in *Proceedings of 7th International Technology, Education and Development Conference*, (Valencia (Spain)), January 2013.

[9] S. H. Edwards and Z. Shams, "Comparing test quality measures for assessing student-written tests," in *Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 354–363, ACM, 2014.

[10] A. Barana, M. Marchisio, and S. Rabellino, "Automated assessment in mathematics," in *Proceedings of COMPSAC*, 2015.

[11] S. Zhigang, S. Xiaohong, Z. Ning, and C. Yanyu, "Moodle plugins for highly efficient programmin courses," in *Moodle Research Conference*, vol. 1, pp. 157–163, 2012.

[12] C. Sanchez, O. Ramos, P. Márquez, E. Martí, J. Rocarias, and D. Gil, "Automatic evaluation of practices in Moodle for self learning in engineering," *Journal of Technology and Science Education*, vol. 5, no. 2, 2015.

[13] S. T. Deane, *Further development on the Moodle CodeHandIn Package*. PhD thesis, Flinders University–Adelaide, Australia, 2014.

[14] R. Lobb, "Coderunner documentation (v2.4.2)." http://coderunner.org.nz/mod/book/tool/print/index.php?id=50, October 2015.

[15] R. Likert, "A technique for the measurement of attitudes.," *Archives of psychology*, 1932.

[16] M. B. Miles, A. M. Huberman, and J. Saldaña, *Qualitative data analysis: A methods sourcebook*. SAGE Publications, Incorporated, 2013.

[17] S. Robertson, "Moodle on the move." WebEx session, https://onlinevideo.napier.ac.uk/Play/5081, 2015.

[12]http://carisma.umlsec.de