Experience Report on an Inquiry-Based Course on Model Checking

Sebastian Krings
Universität Düsseldorf, Hochschule Niederrhein
krings@cs.uni-duesseldorf.de
Philipp Körner, Joshua Schmidt
Universität Düsseldorf
{koerner,schmidt}@cs.uni-duesseldorf.de

Abstract

The development and improvement of model checkers for the validation of hard- and software is an ongoing research topic in computer science. Model checking research connects theoretical and practical aspects; new algorithms are often implemented inside well-known model checkers which have been in development for many years. This is seldom taken into account by university courses on the topic, which often remain on the theoretical level. Hence, they do not provide practical access to the topic for reasons such as lack of time or inaccessibility of tools and their source code.

In this article, we present a recent revision of our course on model checking, shifting from a classical lecture-based format to inquiry and research-based teaching. We document course development, present some didactic methods used and evaluate the course based on peer review and student feedback. Furthermore, we try to assert student engagement empirically.

Introduction and Motivation

The development and improvement of model checkers (Clarke u. a., 1999) for the validation of hard- and software is an ongoing research topic in computer science (Grumberg u. Veith, 2008). Model checking research connects theoretical and practical aspects; new algorithms are often implemented inside well-known model checkers which have been in development for many years. This is seldom taken into account by university courses on the topic, which often remain on the theoretical level, not providing practical access for various reasons.

In particular, both complexity and code volume of commonly used model checkers prevent students from coming to grips with internal workings. In consequence, typical courses on model checking stay on a theoretical level. Students often have no way of experiencing the practical aspects of developing a model checker. This leads to shortcomings in different areas:

- Learning results might be reduced due to missing hands-on experience.
- Regarding bachelor's and master's theses the scope of topics is limited, as students have not learned how to appropriately address practical problems in model checker development.
- Missing experience in project work, tool usage and working collaboratively have been identified as major areas in which students do not meet expectations from industry (Radermacher u. a., 2014; Radermacher u. Walia, 2013; Tafliovich u. a., 2015). Those skills could be acquired en passant in a programming project.

To improve, we decided to remodel our course on model checking. The overall goal was to move from classic lectures to active learning techniques for an improved hands on experience. Furthermore, we noticed that students writing theses at our chair sometimes lack the required knowledge about how research is performed and thus need close supervision and initial training. In order to motivate individual research and to enable students to train their skills, we decided to move from a classic lecture setting to inquiry-based learning. In particular, we intended for the course to follow the typical pattern of asking appropriate questions for research, finding evidence or creating it through experiments, compare and discuss results as well as explain and publish differences discovered.

During the course, participants should:

- Acquire the theoretical foundations of model checking by identifying and analyzing common software errors.
- Align these foundations with the body of knowledge.
- Design and implement a novel model checker as independently as possible.

Barron et al. (Barron u. a., 1998) identified four important aspects contributing to the success of inquiry-based courses:

- Selecting appropriate learning goals,
- Begin with problem-based learning before gradually switching to project work,
- Enable self-assessment and revision,
- Develop an atmosphere and social structures that support participation.

Following, we discuss how we realized these aspects in our course.

Increasing practical relevance and self-responsibility aside, the course redesign serves another purpose. By implementing a novel model checker together with the students we rely less on locally developed tools and avoid tailoring towards in-house techniques. Instead, a course like the one outlined can help evaluating alternative technologies together with our students.

Additionally, the course provides other more long term benefits to the chair. First, we can use the resulting software in bachelor's and master's theses, given that it is more easily accessible than ProB (Leuschel u. Butler, 2008, 2003), the model checker developed at our chair at the University of Düsseldorf. In particular, it is written in Java, which most students know from their undergraduate studies, whereas ProB is written in Prolog, a language not as common. Second, it can serve as a playground for trying out more experimental algorithms or backends.

Following, we start with introducing setting and context of the course. Afterwards, we discuss goals and challenges of remodelling the course using an inquiry-based format. In particular, we document preparation and execution phases. An evaluation of student motivation and engagement will be performed as well. Evaluation and our conclusions lead to another iteration of the course which we describe later on, followed by overall conclusions about both iterations.

Course Setting and Context

Our lecture on model checking is aimed at master students of computer science. In particular, the lecture is part of a major in software engineering and programming languages. In consequence, it should enable participants to immediately continue with a masters thesis in model checking.

There is a corresponding lecture on *Safety Critical Systems*, focussing on how to write software specifications and *use* model checkers and provers rather than implement them. The two lectures are often attended one after the other. However, since we do not enforce a particular order, *Safety Critical Systems* can not be considered a precondition for the course documented in this paper. For both courses, attendance is not compulsory.

Learning Objectives

The learning objectives were already predefined and aligned with the overall curriculum. In consequence, we did not intend to change the high-level goals of the course. After attending, students should be able to

- present and compare different techniques for program verification,
- be able to summarize selected scientific literature on program verification and be able to criticize where appropriate,
- write their own specifications and evaluate them,
- decide on appropriate formalisms, algorithms and tools for given verification tasks.

We believe these goals match the requirements stated by Barron et. al. (Barron u. a., 1998).

Former Course

The former course on model checking before the year 2017 was held in a classical lecture-based format. The lectures dealt with the theoretical foundations of model checking. Lectures were accompanied by weekly exercises used to practice. There were no mandatory submissions but of course students were encouraged to work on the exercises independently.

The course heavily relied on theoretical aspects of model checking, i.e., several theorems and lemmas were proven in the lectures while others had to be proven by the students in the exercises.

Students were not supposed to implement any algorithms throughout the course. Instead, the most important algorithms were discussed in pseudo-code within the lecture. Moreover, the students were occasionally asked to create pseudo-code algorithms for simple tasks on their own such as an explicit-state model checking algorithm for invariant checking.

Students were not introduced in detail to any formalism for writing specifications but worked on an abstract level of formal modeling. However, the students have seen some small models in the lecture without paying any attention to the used formalism but rather focussing on the overall structure. Apart from this, the course remained theoretical neither offering any hands-on experience nor addressing issues

that might arise when implementing model checking algorithms.

In retrospect, we think that the majority of the students suffered from this lecture-based format, as it was not motivating them to work on more advanced topics of model checking. We think that focussing on the theoretical aspects of model checking without any hands-on experience might be daunting for students to work on that topic at all and hinders them to use model checking techniques in future projects. These insights have lead us to remodel the course in the next semester, introducing more interactive elements and practical experiences as we will outline below.

Remodeled Course

In the following sections, we will describe the remodeled course in detail, starting with the participants, the goals and milestones of the redesign and the steps performed in preparation, execution and postprocessing. Additionally, we will discuss two teaching methods employed during the course.

Participants

The first iteration of our remodeled course took place in the summer term of 2017. There were 11 students attending, with 6 of them attending regularly. Three students were undergraduates pulling up courses.

Goals

First, students should gain more practical access to model checking and how it connects to software engineering. Furthermore, the course should be connected to recent research results.

In addition to teaching model checking, we want to encourage independent research. In particular, this includes supporting students throughout the common research phases from finding a suitable hypothesis to publication.

Target Milestones

The target milestones are divided in three stages:

1. Planing and preparation.
2. Implementation and supervision of the research project.
3. Publication and presentation of results.

Individual milestones are given in Table 1. The third phase is not completed as of yet: While we ensure reproducibility for upcoming terms, research results have not been published. This is mostly due to dates and deadlines of appropriate software engineering conferences. As a successful publication cannot be guaranteed, it is not an essential part of our lecture and will not be considered further.

So far, three of the six regularly participating students are willing to contribute to a research paper. We will use this opportunity to explain the rules of good scientific practice as well as the common parts of the scientific publication process, e. g., peer-reviewing.

Challenges

Switching from a classic lecture to inquiry-based learning result in a number of challenges:

- Cognitive requirements are higher, as we switch from knowledge reproduction to production.
- Course progression is less linear. In particular, project progress will be fluctuating and has to be monitored closely to allow for timely interventions in case goals might become unreachable otherwise.
- Adding an additional programming task to the curriculum increases the individual workload students are exposed to. In consequence, individual motivation and commitment has to be increased.
- Research has to be controlled in order to avoid getting off track. Students should have as much freedom as possible, while still being guaranteed to reach the intended learning outcomes.
- Due to the higher amount of group work and cooperation, exams have to be prepared carefully to meet both the didactic requirements and the ones posed by exam regulations.

Documentation Preparation

As stated in Table 1, we began course preparation by collecting and inspecting existing lecture material such as slides and exercise sheets. Furthermore, in order to gain additional material, the latest literature on model checking was sighted as well.

Reducing existing material to make room for the programming tasks proved difficult for two reasons:

- It was hard to anticipate the speed with which students would progress. As this is our first inquiry-based attempt at teaching model checking, we could not rely on previous experiences.
- As stated above, we intended for the module to be as open as possible. In particular, we wanted to avoid predetermining techniques and algorithms to be used. However, this made it impossible to anticipate what the students would come up with first. We needed to prepare for several possible pathways through the course.

To reduce overhead, we decided to reduce content even more, until we reach the bare minimum of model checking knowledge. Additionally, we prepared several collections of articles, slides and exercises for different subtopics. As soon as speed and direction of student research can be asserted, we could add or remove collections as needed.

Table 1: Phases, Milestones and Time for Completion

	Preparation	
1.	Inspection of existing material	5 h
2.	Literature research and selection, collect relevant papers and documentation to be supplied to students	5 h
3.	Didactic reduction: make room for programming tasks by reducing course content without omitting learning outcomes	5 h
4.	Prototypical implementation to assert difficulty level and needed effort, identify obstacles	10 h
	Execution	
5.	Project and time management, task distribution, team building (done in lectures)	13 h
6.	Discussion, monitoring of results (outside of lectures)	15 h
7.	Further implementation work, bugfixing (outside of lectures)	13 h
8.	Peer review by didactics department	6 h
	Postprocessing	
9.	Collecting results, documentation	10 h
10.	Benchmarks and performance evaluation	10 h
11.	Publication process	10 h
12.	Evaluation and preparation for upcoming terms	5 h
	Total	107 h

To assert the overall workload and to identify possible obstacles, we developed a prototypical implementation of a model checker suitable for the course. By doing so, we learned that developing a parser for B, i.e. a software that turns the human readable representation of software into a machine readable form, is a more time-consuming task than we initially suspected. Furthermore, we all had attended lectures on *compiler construction* and thus had at least theoretical knowledge in writing parsers. However, as stated above, we could not assume the same of the participating students.

In consequence, we decided to make our prototypical parser available for the students instead of making it a part of the programming project. This allowed us to focus more on model checking itself, rather than spending time on infrastructure. While this somewhat reduces possible learning outcomes, we still feel our decision is justified. This is stressed by the fact that several other lectures, e. g., *compiler construction*, *dynamic programming languages* or *logic programming*, feature the development of parsers.

Documentation Execution

In the following section, we will document our experience in course execution. In particular, we will discuss how we introduced the topic and helped students formulate initial research questions. Following, two key methods used throughout the course are discussed. We conclude with our approach to grading.

Developing a Research Hypothesis

Following the work by Barron et. al. (Barron u. a., 1998), we decided to start our course problem-oriented instead of immediately confronting students with the need to develop a research question themselves[1]. To do so, we followed a four-step approach to introduce model checking:

- Error- and hazard analysis of an elevator control software,
- Introduce specification languages, by discussing a reduced version of the B language (Abrial, 1996).
- Collaboratively develop key definitions such as state spaces and transitions,
- Collaboratively develop a simple explicit-state model checking algorithm.

We decided to rely on B as an input language, mainly because it is the specification language most commonly used at our chair. However, it is a rather complex language with many features students had to learn. In retrospect, a simpler language could have been used without reducing learning outcomes. In consequence, the course was modified for later iterations, as we will discuss later on.

In the following sessions, we switched to a more inquiry-based approach: For instance, the reduced input language did not include a way to specify certain system properties. As students were writing their own software models in order to gain test cases, those shortcomings became obvious. In consequence, students had to come up with language extensions and develop new model checking algorithms to verify them.

[1] For a comparison on inquiry-based and problem-based learning see, for instance, (Oguz-Unver u. Arabacioglu, 2014).

Of course there are different extensions to simple explicit-state model checking. We intended for the students to find them independently as well as share and discuss their findings. To avoid errors, we had four lectures allocated to scientific foundations. In summary, those were barely needed, as our students were able to independently discover and provide foundations. Without further guidance, they managed to do literature collection and research and were able to draw appropriate conclusions for further development. Furthermore, they managed to bridge the gap to their undergraduate studies where needed autonomously.

Teaching Methods

In the following sections, we will discuss two teaching and organization methods, which we used throughout all lectures. The two methods complement each other: The hazard collection helps keeping in mind the big picture. In contrast, the Kanban board is used to structure the programming project into parts and helps to discover and tackle todos in an organized manner.

Collection of Hazards

To motivate the how and why of model checking, we began the first lecture with a simplified hazard analysis. Working in groups, students were supposed to describe behavior scenarios of an elevator that performs as bad as possible. Scenarios were collected and sorted, first by grouping corresponding ideas such as “the elevator never opens its doors” and “the elevator never closes its doors”. The resulting collection is shown in Figure 1a.

Not all of the identified scenarios can be prevented using a model checker. We thus supposed to reorganize our collection by two axes: The dangerousness of the scenario and the extent to which the elevator control software is involved. The sorted collection is shown in Figure 1b. In the upper right corner, we see those scenarios that should be considered first, i. e., those that are hazardous and primarily caused by software defects.

We keep the collection around during the course of the semester, removing those cards representing errors our model checker was able to detect. In consequence, the hazard collection was driving the inquiry-based learning process: While some errors were easy to detect using simple explicit-state model checking techniques, others made more involved techniques necessary. For instance, a property such as “If the elevator is moving, doors are closed” is comparably simple to verify. In contrast, “At some point in time the elevator is moving”, is more involved due to time-based reasoning.

Project Management using Simplified Kanban Boards

To manage programming tasks and how they are distributed between the participants we used a method based on simplified Kanban boards. Every occurring task is collected on a (virtual) flashcard. While being processed, tasks follow a predefined live cycle by moving from and to different stacks:

1. Backlog - Newly created tasks start here. If a task is not actionable, because a precondition is missing, a new flashcard for the precondition is added. If a task is too large or to unwieldy to handle, it is split into multiple subtasks. Tasks that can be approached now are moved to the next stack.

2. To Do / Ready - Includes actionable tasks that can be addressed right now. Students can pick a task and assign it to themselves. Once this happens, the task moves forward.

3. In Development - Tasks in process by someone. Each task in this stack has someone responsible for its handling assigned.

4. Testing / In Review - Task that the assigned students consider finished are moved here. For quality control, another student has to verify whether the task has been handled satisfactorily. This is usually done by adding automatic test cases and by performing manual code review. If problems are identified, the task is moved back to “In Development”. Added test cases and comments should now help find a better solution. Otherwise, the task is moved to the next stack.

5. Done - Tasks that have been finished and verified.

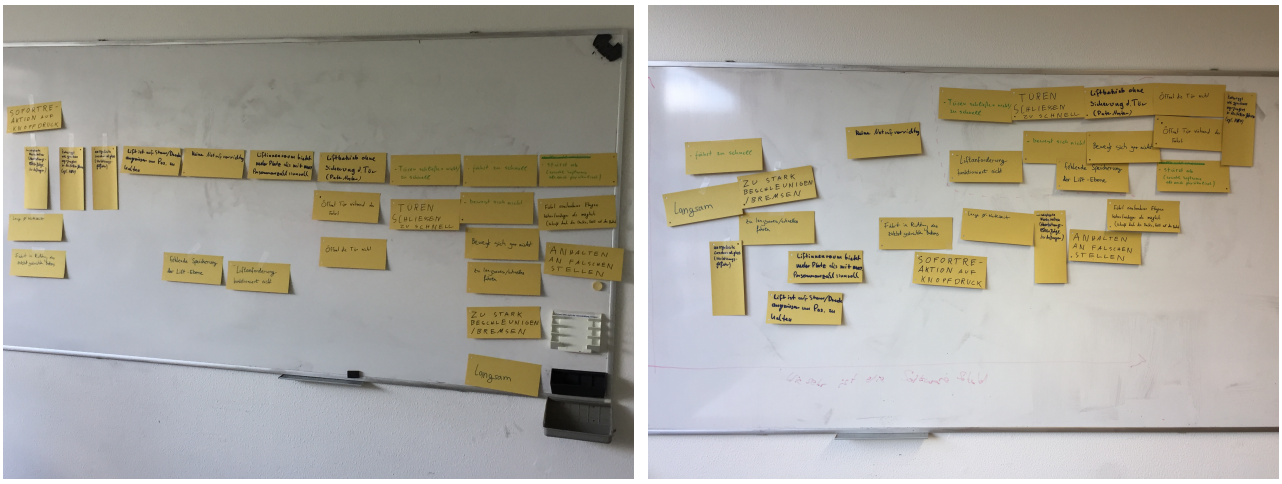
We used a virtual board on GitHub as shown in Figures 2 and 3. The method proved easy to learn for students and efficient in handling programming process. Furthermore, it served as a documentation of participation that could be considered for grading, as we will discuss in the following section.

Grading

As the course was teaching both theoretical and practical aspects of model checker development, exams were supposed to measure both parts. In additional, we had to ensure that grading complies with the examination regulations. In particular, regulations enforce grades to be based on individual performance, i.e., group work can not easily be taken into account.

To improve constructive alignment (Biggs, 1996), we implemented a combination of formative and summative exams:

- Constant participation in the programming project was documented using the Kanban project management method as outlined above. As changes in tasks and issues were recorded together with the student’s username, following the individual contribution was easy. This formative part mostly considered the individual contributions to the programming project and takes into account that different parts of the model checker were developed



(a) By Topic (b) By Hazardousness / Software Influence

Figure 1: Elevator System - Hazard Analysis

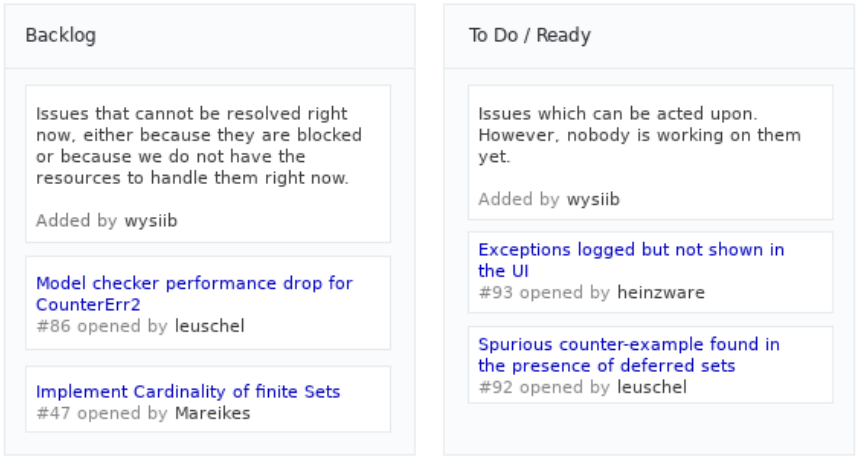


Figure 2: Kanban Board, Part 1

independently. For example, UI and model checking core were developed by different students and at different times.

- Changes in attitude, development of soft skills such as ability to cooperate and stance towards ethical issues in software development could be observed both in the live sessions and the online discussions. Of course, those cannot simply be graded on observation alone and thus mostly served for the lecturers to assert progress.

- Theoretical foundations of model checking were verified by a written summative exam at the end of the semester. Keep in mind that we had to reduce theoretical content to be able to fit in the programming project.

The combined exam is able to verify, whether the learning objectives stated above have been reached:

- The goal to enable the students to present and compare different techniques for program verification was verified using project contributions. Students had to decide between different algorithms (i.e. compare) and they had to present the implementations during the reflection & evaluation sections.

- The goal to enable the students to summarize selected scientific literature on program verification and be able to criticize where appropriate, was partially covered by the reflection & evaluation sections. Additionally, we used the exam to ask students to select an appropriate algorithm for given problems, i.e. to criticize weaknesses ruling out algorithms.

- Students wrote their own specifications and evaluated them in order to create test cases for the programming project.

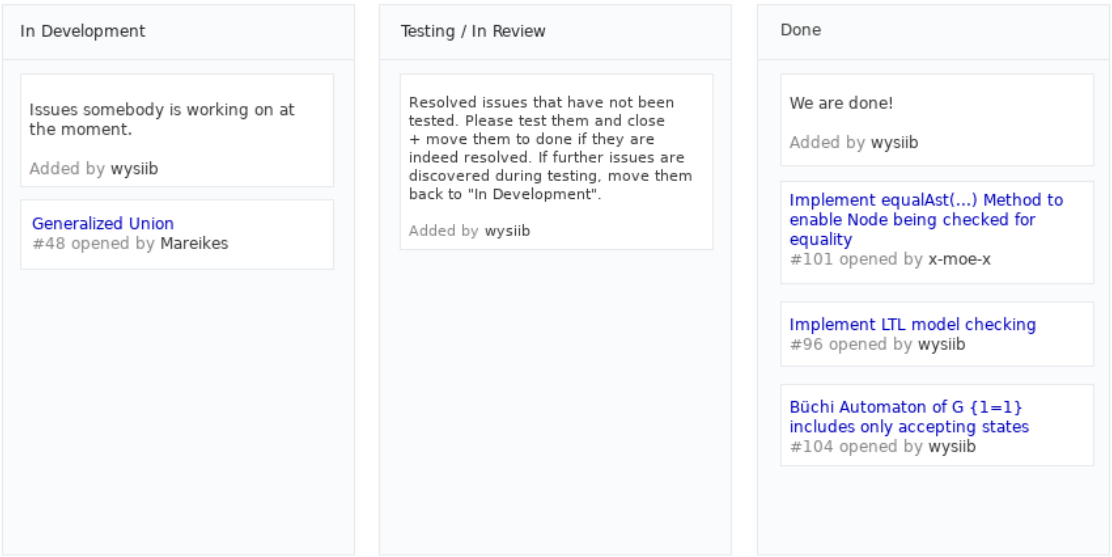


Figure 3: Kanban Board, Part 2

- Again, we used the written exam to ask students to decide on appropriate formalisms, algorithms and tools for given verification tasks.

Course Evaluation

This section will discuss several means of evaluation we employed to measure the success of the redesign. We will discuss review by other faculty members as well as student feedback and engagement.

Peer Review

Two sessions were monitored and reviewed by other faculty members and members of the didactics department. Each time, certain weaknesses were spotted and later fixed based on suggestions of the observers.

R & D Session

The intention of the first observed session was to develop novel model checking techniques in order to be able to detect more of the hazards collected in Figure 1b. In particular, we tried to encourage research and experimentation and avoided discussing known solutions beforehand. We decided to have the session supervised in order to verify if we hit the sweet spot between student research and lecture progress.

The overall concept of the sessions and the belonging teaching methods were evaluated as coherent and aligned with the goals of the sessions. However, weaknesses were spotted when it comes to student interaction and result summarization and recapitulation:

- Sometimes, we failed to ensure that all students had understood the currently discussed idea well enough to participate in further discussions. In consequence, some ideas for new algorithms were developed within smaller groups of students and only later discussed with the group as a whole. For future sessions we decided to discuss and document new ideas more thoroughly, including visualizing them on a flip chart. Resulting documentation should also help later implementation.

- For some ideas the group decided to be worthwhile of further experimentation, the next steps to take remained unclear. In particular, resulting tasks have to be identified immediately and clear commitment to resolving them has to follow.

Reflection & Evaluation Session

The second session that was observed was meant as a synchronization point between different groups working individually. While certain students were working on verification techniques for infinite software models, others were working on time-based reasoning. As discussed, both topics are considered essential for the course. We thus decided to form focus groups with the intent of later updating the other groups.

Furthermore, presenting intermediate results to other groups was intended to account for the appropriate reflection upon the executed research tasks. As pointed out in (Blumenfeld u. a., 1991; Schauble u. a., 1995), project-based work could otherwise focus too much on the execution without evaluation results and lessons learned. In consequence, students would have less opportunity for self-assessment and revision, violating one of the principles of successful inquiry-based courses stated by Barron et. al. (Barron u. a., 1998).

During the lecture, presentation of intermediate results was sluggish, students appeared to be only superficially prepared. In retrospect, we identified our imprecise task statement as the most likely reason. Again, we adapted following sessions in order to improve:

- In addition to asking for a presentation of intermediate research results, we supply an outline:
  1. Short presentation of used techniques and how they are supposed to work.
  2. Summary of difficulties and solutions implemented so far.
  3. Open problems, followed by a discussion of possible solutions.
  4. Code examples and actual implementation where sensible.
- To increase student commitment, outline and presentation content has to be discussed with the lecturers beforehand (see milestone 6 in Table 1).

Student Feedback

Due to the small number of participants, no official evaluation was performed by the university. For the same reason, there are no former evaluations we could compare to.

However, during the course, we performed several intermediate evaluations using short online surveys. Feedback was very positive and encouraging. In particular, the course was described as both interesting and challenging at the same time. Furthermore, students evaluated their own learning achievements as high. More sustainable learning was attributed to the increased self-reliance compared to other courses.

The main point of criticism was the volatile speed of progression during the lectures. While this is not uncommon for research-based processes, it turned out to be the major cause of frustration for students. As an immediate remedy, we started to monitor progress more closely and intervened early once students got stuck. For the next iteration of the course, we believe we should reconsider the balance between problem-based and inquiry-based learning.

In order to gain more insight into workload, motivation and learning outcomes, we performed another feedback round when writing this article.

The student's workload was rated with an average of 3.75 on a scale from 1 (= low) to 5 (= high). This is in line with our expactations and does not suggest that the inquiry-based setup increased the overall workload. Learning outcome was rated with an average of 4.375, while overall motivation was rated with an average of 4.625, both again on a scale from 1 (= low) to 5 (= high).

However, given the low number of course participants, we cannot claim any statistical significance.

Grades

In 2017, six students participated in the exam. Without taking into account the project work, all students passed the exam with an average of 1.43 (equivalent to US 3.9 - 3.7, "excellent"). When taking into account the formative part of the exam, i. e., project work and participation, the average improves to 1.28 (equivalent to US 3.9, "excellent").

Table 2: Average Grades

	2014	2015	2016	**2017**	2018
# Students	2	5	7	**6**	5
∅ Grade	1.85	2.58	1.71	**1.28**	1.88

For comparison, the average grades of different course iterations is given in Table 2. The low average in 2015 is due to one student not passing. If only passing students are counted, the average improves to 1.98.

Judging by the average grades, the highly inquiry-based course in 2017 performed better than the lesson-based course in 2016. Furthermore, the next iteration of the course, which includes several improvements we we will discuss below, again exhibits a slightly worse average grade.

Of course, we would like to attribute the improved grades to the practical research experience, that students could gain during the hands-on sessions. However, with only five to seven students participating in the exams, the sample size is much to small to draw reliable conclusions.

Student Engagement

One of the main challenges was to keep students motivated to participate in research and programming. In particular, participation outside of lecture hours had to be ensured. Given that we used Git to record all changes made to the source code, we can use the data collected for statistics on participation and engagement. Apparently, we were able to motivate participants to consistently and autonomously work towards the course goals.

For instance, this can be seen in Figure 4, showing a so called "punchcard". Time of day on the x-axis and days of the week on the y-axis form a grid in which dots of different diameters are drawn. The larger a dot is, the more extensively the source code was changed at that time interval.

Of course by far the biggest amount of changes was made during lectures, exercises and tutorials, wednesday and friday 12 - 2 p.m. While these are the most active phases, we can see student activity throughout all days and times. Particularly pleasant is the constant activity occurring wednesdays at 7 p.m. and fridays at 8 p.m. Apparently, they are caused by students revising lecture material and incorporating it into the programming project.

Another indicator for high motivation is that the project was both fascinating and motivating enough to cause the occasional all-nighter. For instance, we see source code changes thursdays at 3 a.m. and saturdays at 4 a.m. To determine if this is an indication of motivation or high work load, we asked the students

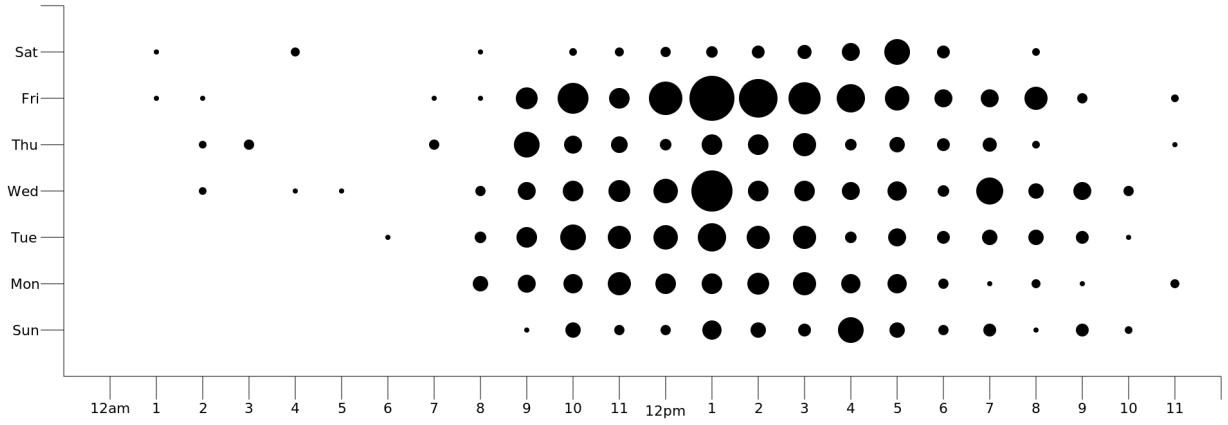


Figure 4: Source Code Changes per Day / Time

to evaluate. Students stated that late night work was not due to high work load or last-minute submissions, but to individual interest, motivation and time management. As we already stated, the workload was rated as 3.75 on average, using a scale from 1 (= low) to 5 (= high).

With the data depicted in Figure 5 we tried to evaluate, whether students were indeed following an inquiry-based approach to the course topics. To do so we tried to evaluate if progress was linear or if students had to follow the typical research pattern of finding approaches, try and experiment and later evaluate. The x-axis shows project progress from February to September. Y-axis shows additions and deletions performed in the source code, i.e., the higher the green part, the more source code has been added to the project. The red part depicts deletions, that is the lower the red bar is, the more source code has been remove from the project.

The small peak at the begin of February is caused by the lecturers developing a prototypical implementation as we discussed for the preparation phase. Lectures itself started in april. In summary, the graph shows that a considerable proportion of the source code added has been removed later on. The amount of lines removed rises and falls with the amount of lines added. This hints at the fact that students did not only add new code once a new idea was developed. Rather, constant reevaluation lead to code being revisited or even replaced. Participants did not progress in a linear fashion.

Another interesting point is the peak mid of June. At that time, students developed certain ideas of "symbolic model checking" and started to implement them in our tool. For this peak, there is no following peak with deletions. This could mean that students made some fundamental progress by reinventing and realizing a classic idea of model checking.

Publication

The last state of inquiry and research, namely publication, was not part of the course for several reasons. First of all, preparing a meaningful publication takes a lot of work and publication cycles can be quite time consuming as well. Thus, it was impossible to fit publishing into the course as well. Second, while publishing teaches a lot about how the scientific community works, it does not contribute to the course topic.

Nevertheless, three of the participating students were interested in writing a follow-up paper and presenting their work to the community. This gave us the opportunity to introduce them to the common publication process, including concepts such as peer review as well as pre- and post-proceedings of conferences.

The overall process went through a number of meetups and discussion sessions:

1. In a first session, we discussed the publication process as it usually takes places.

2. For two sessions, we discussed how to write an interesting paper, mostly following Simon Peyton Jones' advise on the topic (Jones, 2018).

3. We did a brainstorming session in order to identify the key research questions students answered during the course and to assert which of them would be relevant to a larger audience.

4. We had several meetups to discuss the writing progress and to synchronize us.

As the publication process was not part of any official lecture or university event, we were unable to provide credit points to the participants. Yet, all three of them were very enthusiastic about the idea of writing their first article. In retrospect, we believe that it was mostly the fact that we were aiming for a "real scientific workshop", that was highly motivating. Students felt as part of the scientific community and truly as peers among peers.

Figure 5: Source Code Add / Delete by Date

To our joy, the paper (Petrasch u. a., 2018) was accepted for the *18th International Workshop on Automated Verification of Critical Systems*, which took place in Oxford, GB in January 2018. Two students attended the workshop, with one of them giving the presentation. It lead to fruitful discussions both about the topic and about the course itself. As many of those attending were teachers themselves, success reports of our hands-on approach were of particular interest to them.

Lessons Learned - the Next Semester

Two terms after we remodeled the course, the next iteration took place. During the preparation stage, we started reconsidering the course and how to develop it further. In particular, we intended to address the following concerns, some of which occurred in the old course, while some are caused by environmental variables:

- We do not think the presented approach scales to larger groups. A single project for all students seamed infeasible, since 18 students registered and even more showed up in the first session.
- We were concerned that knowledge did not propagate evenly, as students specialized on a subset of topics, e.g. LTL or explicit-state model checking.
- As the B language is a rich formalism, many problems unrelated to model checking might surface.
- Given that theoretical foundations were reduced quite heavily, we feared students working on more advanced topics had to catch up individually.

In consequence, we tried to find a middle ground between focussing on the implementation of a model checker and a lessons-based course.

As knowledge propagation was our most prominent concern, we decided early on that each student should implement a model checker on their own. This is a trade-off between allowing exploration and hands-on experience as well as feature-completeness, as obviously a single student is not expected to do the work that six students did beforehand.

When switching from group projects to individual programming tasks, students do not gain experience in best practices of software development in a team and project management. However, we think that this was more of a side-effect, given that collaborative software development and synchronization in teams is taught in other courses.

Thus, the overall programming workload had to be reduced. To do so, we introduced two changes. First, we changed the formalism from B to petri nets (Petri, 1966), which are considerably easier to understand and implement: As far as the language interpreter is concerned, only addition and subtraction is required. This allows to focus on model checking techniques instead of writing a more complex language interpreter.

However, writing meaningful test cases is harder in such a low-level formalism. Therefore, we used benchmarks from the annual model checking contest (Kordon u. a., 2016) instead of making students writing their own. As in the previous iteration, we provided a parser for the input format.

Furthermore, we split the programming project into two parts. In the first part of the project, the students had to implement a very basic explicit-state model checker to check for deadlock freedom and to verify

certain safety properties. This part of the project had to be passed in order to receive the permission to write the final exam. The second part of the project dealt with more advanced topics: The existing model checker had to be be extended in order to verify linear-time properties by implementing algorithms that have been introduced in the lecture. We graded the second part of the programming project of each student and incorporated the result into the overall grade with a weight of 25 %.

Secondly, we brought back some of the theoretical lessons and exercises, yet cutting down on enormous proofs so there was enough time to spare to let our students work on their model checkers. We intended that the theoretical exercises in particular deepen the knowledge and allow pointing out connections between individual components throughout the lectures.

Thirdly, in addition to the parser, we provided some components of the model checker that were not covered in the lecture and required cumbersome implementation of algorithms that are only sparsely documented in literature.

In the exam, we focused on theoretical problems instead of applying basic algorithms since each student implemented them already. Thus, given that we think the exam was harder overall, we are pleased with the result in 2018 as shown in Table 2.

During supervision and discussing of the model checking projects, it became clear that many students procrastinated and put off work until the last few weeks, not making use of the available time given by cutting back lectures. This might be caused by the lack of fruitful discussion about how components should be implemented that results in inspiration to try it out and peer-pressure to reach common goal during the next week. A possible solution might be to enforce peer-programming during practical sessions.

Conclusion

In summary, we believe that the restructured course on "model checking" met our self-set goals. Realization was more hassle-free than we anticipated.

In particular, contrary to our concerns, we had no problems motivating students to active participation. Even though the course was quite experimental at times, everybody was willing to try. We only had to intervene with speed and progress direction seldomly.

One of the key challenges proved difficult however. Due to the scale of the programming projects, students had to form groups concerned with different partial aspects. For instance, some were working on techniques for the verification of temporal properties while other were working on performance improvements. However, all students were supposed to take the same exam.

As planned, we had regular meetings dedicated to synchronizing the different groups. However, specialist knowledge remains and is not distributed equally throughout the participants. While the exam and its results show that all participants reached the desired learning outcomes, we think that with a better focus on knowledge transfer an even better outcome could have been reached.

To summarize, our refurbished lecture on *model checking* is gradually being refined. It is intended to replace the former lessons-only course in following iterations.

We believe that a few insights can be carried over from our course to other courses. In particular, we believe that a strong connection between teaching and research is highly beneficial and can be achieved with little overhead using programming projects. Depending on the scenario, a switch to inquiry-based learning is feasible and leads to high motivation among participants, without causing too much reduction in course content. However, focus has to be given to the synchronization of participants, i.e., teachers have to ensure that nobody is left behind.

Last, our course shows that with the right motivation and proper support, inquiry-based learning methods can produce research results on a very high level.

Acknowledgments

The authors would like to thank the university didactics department of the Heinrich-Heine-University for their constant support and consulting. In particular, the authors would like to thank Susanne Wilhelm for the project supervision and Jens Bendisposto and Janine Golov for peer reviewing the lectures.

References

[Abrial 1996] ABRIAL, J.-R.: *The B-book: Assigning Programs to Meanings*. New York, NY, USA : Cambridge University Press, 1996

[Barron u. a. 1998] BARRON, Brigid J. ; SCHWARTZ, Daniel L. ; VYE, Nancy J. ; MOORE, Allison ; PETROSINO, Anthony ; ZECH, Linda ; BRANSFORD, John D.: Doing With Understanding: Lessons From Research on Problem- and Project-Based Learning. In: *Journal of the Learning Sciences* 7 (1998), Nr. 3-4, S. 271–311

[Biggs 1996] BIGGS, John: Enhancing Teaching through Constructive Alignment. In: *Higher Education* 32 (1996), Nr. 3, S. 347–364

[Blumenfeld u. a. 1991] BLUMENFELD, Phyllis C. ; SOLOWAY, Elliot ; MARX, Ronald W. ; KRAJCIK, Joseph S. ; GUZDIAL, Mark ; PALINCSAR, Annemarie: Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. In: *Educational Psychologist* 26 (1991), Nr. 3-4, S. 369–398

[Clarke u. a. 1999] CLARKE, Edmund M. Jr. ; GRUMBERG, Orna ; PELED, Doron A.: *Model Checking*. Cambridge, MA, USA : MIT Press, 1999

[Grumberg u. Veith 2008] GRUMBERG, Orna (Hrsg.) ; VEITH, Helmut (Hrsg.): *25 Years of Model Checking: History, Achievements, Perspectives*. Berlin, Heidelberg : Springer-Verlag, 2008

[Jones 2018] JONES, Simon P.: *How to write a great research paper*. https://www.microsoft.com/en-us/research/academic-program/write-great-research-paper/, 2018. – Accessed: 2018-10-24

[Kordon u. a. 2016] KORDON, Fabrice ; GARAVEL, Hubert ; HILLAH, Lom M. ; PAVIOT-ADET, Emmanuel ; JEZEQUEL, Loïg ; RODRÍGUEZ, César ; HULIN-HUBARD, Francis: MCC'2015 – The Fifth Model Checking Contest. (2016), S. 262–273

[Leuschel u. Butler 2003] LEUSCHEL, Michael ; BUTLER, Michael: ProB: A Model Checker for B. In: *Proceedings FME'03*, Springer, 2003 (LNCS 2805), S. 855–874

[Leuschel u. Butler 2008] LEUSCHEL, Michael ; BUTLER, Michael: ProB: an automated analysis toolset for the B method. In: *Int. J. Softw. Tools Technol. Transf.* 10 (2008), Nr. 2, S. 185–203

[Oguz-Unver u. Arabacioglu 2014] OGUZ-UNVER, Ayse ; ARABACIOGLU, Sertac: A comparison of inquiry-based learning (IBL), problem-based learning (PBL) and project-based learning (PJBL) in science education. 2 (2014), 07, S. 120–128

[Petrasch u. a. 2018] PETRASCH, Jessica ; OEPEN, Jan-Hendrik ; KRINGS, Sebastian ; GERICKE, Moritz: Writing a Model Checker in 80 Days: Reusable Libraries and Custom Implementation. In: *ECEASST* (2018)

[Petri 1966] PETRI, Carl A.: *Communication with automata*, Universität Hamburg, Diss., 1966

[Radermacher u. Walia 2013] RADERMACHER, Alex ; WALIA, Gursimran: Gaps Between Industry Expectations and the Abilities of Graduates. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. New York, NY, USA : ACM, 2013 (SIGCSE '13), S. 525–530

[Radermacher u. a. 2014] RADERMACHER, Alex ; WALIA, Gursimran ; KNUDSON, Dean: Investigating the Skill Gap Between Graduating Students and Industry Expectations. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA : ACM, 2014 (ICSE Companion 2014), S. 291–300

[Schauble u. a. 1995] SCHAUBLE, Leona ; GLASER, Robert ; DUSCHL, Richard A. ; SCHULZE, Sharon ; JOHN, Jenny: Students' Understanding of the Objectives and Procedures of Experimentation in the Science Classroom. In: *The Journal of the Learning Sciences* 4 (1995), Nr. 2, S. 131–166

[Tafliovich u. a. 2015] TAFLIOVICH, Anya ; PETERSEN, Andrew ; CAMPBELL, Jennifer: On the Evaluation of Student Team Software Development Projects. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. New York, NY, USA : ACM, 2015 (SIGCSE '15), S. 494–499