Increasing Student Self-Reliance and Engagement in Model-Checking Courses

Philipp Körner[1] and Sebastian Krings[2]

[1] Heinrich Heine University, Universitätsstraße 1, 40225 Düsseldorf, Germany
p.koerner@hhu.de
[2] Independent Researcher
sebastian@krin.gs

Abstract. Courses on formal methods focus on two aspects: teaching formalisms and exemplary applications as well as teaching techniques for implementing tools such as model checkers.
In this article, we discuss the second aspect and typical shortcomings of corresponding courses. As courses often focus on theoretical results, opportunities for working on real implementations are scarce. In consequence, students are easily overwhelmed with transfer tasks, e.g., when working on existing model checkers during theses or research projects.
We present several iterations of our course on model checking, including their goals, course execution as well as feedback from peers and students. Additionally, we discuss how the Covid-19 epidemic impacted our course format and how it was made more suitable for online teaching.
Finally, we use these insights to discuss the influence of formality on student engagement, and how to incorporate more practical aspects by introducing inquiry and research-based teaching.

Keywords: Education· Model Checking · Inverted Classroom · Experience Report.

1 Introduction

The development and improvement of model checkers [9] for the validation of hard- and software is an ongoing research topic in computer science [12]. 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. Thus, they typically have large and often involved code bases posing an entry barrier for students.

This is seldom taken into account by university courses, which often remain on the theoretical level, not providing practical access for various reasons. The same used to hold true for our approaches to teaching model checking.

Different shortcomings of typical courses on formal methods and model checking have also been identified in a whitepaper published at FMFun [8]. Among other reasons such as limited exposure to formal methods and missing integration of FM courses with the rest of the curriculum, the whitepaper lists a number of shortcomings relevant to course design and execution:

- FM courses are usually very formal, especially when compared to the more hands on example-drive approach to teaching programming languages.
- FM courses often provide only limited practical experience.
- Initial interest is low, formal methods are perceived as inaccessible.
- Students are often unable to bridge the gap between theory and tools.
- A focus on technical details might distract students from key learning goals.

With our course on model checking, we try to overcome these restrictions by increasing student involvement and motivation. This is done by providing practical experience in developing, testing and using a model checker and abstracting away from one concrete formal method.

We would like to add, that project experiences, tool usage and working collaboratively have been identified as major areas in which students do not meet expectations from industry [22,21,25]. Those skills could be improved en passant when integrating practical (programming) aspects into an FM course.

The rest of the paper is structured as follows: First, we briefly present the context of our course on model checking in Section 2. Afterwards, we outline several couse iterations in Section 3, including their goals, course execution and feedback from peers and students. We focus especially on the latest iteration that shifted the course to a more suitable format for online-teaching, as necessary during the COVID-19 epidemic. After comparing the average grades for different iterations and trying to explain them in Section 4, we finally discuss the influence of formality on student engagement, and how to incorporate more practical aspects by introducing inquiry and research-based teaching in Section 5.

2 Context

At the Heinrich-Heine-University Düsseldorf, we teach two master's courses — each worth 5 ECTS — concerning formal methods: Firstly, "safety-critical systems", where modeling and validation of B [1] and Event-B [2] specifications is taught. This is done using both the animator and model checker ProB [17],as well as the Event-B IDE Rodin [7], Recently, ProB was also integrated into Jupyter notebooks [11], which can be used for teaching as well.

The second course, which is the main focus of this paper, is "Model Checking". This course roughly follows the first half of Principles of Model Checking (PoMC) [3]. It mainly deals with algorithms and techniques for implementing model checkers. After course completion, students should be able to unfold transition systems from a specification, classify different kinds of properties (safety, liveness, ω-regular), implement (fair) algorithms for invariant, safety and LTL model checking, and to formulate properties given in natural language in a suitable, mathematical notation. Depending on the lecturer, these two courses may overlap somewhat on the modeling aspect.

In contrast to safety-critical systems, the course on model checking has displayed the typical shortcomings discussed above: interest was low in general (i.e., a small number of enrollments), students attended the course in a strongly passive manner and grades were fluctuating.

3 Course Evolution

In this section, we present several iterations of our course. We highlight the ideas and goals, outline their execution and include available feedback.

Aside from informal ad-hoc feedback, we do not have data on the original lecture-based course. Official course evaluations during the semester require a minimum number of student answers that was not reached.

The first and last course iteration presented have additionally been supervised by the didactics department. This includes reviews by peer lecturers (who may teach in a different faculty) during the planning phase as well as during the execution of the course. The didactics department also requires resilient feedback in written or electronic form.

3.1 The Origins: A Classical Lecture-Based Course

Teaching & Course Execution The baseline course is a "traditional", lecture-based setting with additional exercises. Exercise sheets were handed to the students on a weekly basis and discussed in the following week. Exercises were voluntary and often on a larger scale. A major point of criticism by students was that exercises could neither be solved nor discussed in time. Independent preparation was, nonetheless, expected.

The lecture itself was made up of the more formal and theoretical aspects of model checking: The most important definitions, examples, lemmas and theorems of PoMC [3] were put onto slides and proven. Algorithms were presented in pseudo-code, but students were not given the task to implement them.

No formalism was discussed in detail. Instead, snippets or short models of B, CSP or nanoPromela[3] were used to demonstrate how certain aspects of specifications may look like. Aside from some discussion on the state space explosion problem, neither practical experience, experiments nor implementation concerns were part of the course.

Grading As all activity during the semester was voluntary, grades were given solely based on the performance in a 90 minute written exam. The most common questions tested whether the student was able to extract a transition system from a specification, calculate the handshaking of two transition systems, understand Büchi automata, and check and (dis-)prove equivalences of LTL formulas or extract them from natural language.

Reflection We assume that the original course format is at least partially responsible for the relative unpopularity of the course: The lecture-based format, combined with a strong mathematical component and the resulting reputation to be (too) work-intensive, was certainly reducing motivation for attendance.

[3] A subset of SPIN's [13] input language that is introduced in PoMC

Following the FMFun whitepaper [8], focusing on the mathematical aspects of FM reduces motivation if done before students are able to see the benefits of fully formal approaches. Additionally, given the relative unpopularity of formal methods compared to hot topics such as data science and artificial intelligence [8], only few students enlisted at all.

During the course, students remained unengaged both during lectures and exercise sessions. Without hands-on experience, students cannot witness the benefits of such tooling themselves. Thus, students are unlikely to use or may even be deterred from exploiting model checking techniques after the course.

3.2 First Iteration: Introducing Research- and Inquiry-Based Learning

Idea and Goals Several insights from trainings on teaching and learning made us realize that the model checking course should be reworked in order to increase student interest, engagement and to overcome the shortcomings discussed.

To improve, we decided to remodel our course. 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 lacked the required knowledge about how research is performed and thus needed close supervision and initial training.

In order to motivate individual research and to enable students to train their skills, we moved from a lecture-based setting to inquiry-based learning. In particular, we intended for the course to follow the typical pattern of finding and asking research questions, collecting evidence or creating it through experiments, compare and discuss results as well as explain and publish differences discovered.

Teaching & Course Execution The first iteration and its course execution has already been described in detail [16]. In the following, we thus only give a brief overview. In line with the intended learning outcomes, 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.

To reach the goals, we started with an introductory example. Together with the students, we brainstormed numerous hazards the control software of a lift could suffer from. Following, we used an example-driven approach to introduce the B language. Since students were mainly supposed to learn about model checking algorithms and implement a model checker, technical details on B's semantics were not relevant.

Afterwards, students were asked to describe the behavior of the control software in (their impression of) B. Once consensus on the software specification was reached, students were asked to (independently) research verification algorithms

and to implement them collaboratively. While doing so, students realized what is needed to turn the hazard collection into a checkable specification: invariants, (temporal) logic, etc.

During their research, we had different sessions where students could discuss their findings and ideas with us and where we provided further input and clarification. Additionally, these sessions where used to ensure student research was going into the intended direction.

Grading When planning courses and exams, the learning outcomes, teaching methods and assessment should be constructively aligned [5]: Roughly summarized, student's activities during the course should reflect the intended learning outcomes. Simultaneously, the exam should be similar to those activities as well. Otherwise, students will learn what they think will be needed to pass the exam rather than what the course is supposed to teach them.

Thus, a more practically oriented course needs an appropriate grading method: We decided to use a combination of grading participation in the research and programming projects and a classical exam for the more theoretical parts.

Feedback

Peer review Two separate sessions were monitored and reviewed:

- An R & D session in which the students drove the prototypical model-checker forward by discussion algorithmic approaches and further implementation.
- A session meant as a synchronization point between two groups working on infinite-state approaches to model checking or time-based reasoning.

For the R & D session, the overall concept of individual inquiry and research by students combined with discussing existing approaches once they were "discovered" was seen as innovative, effective and appropriate for the goals we set. However, some weaknesses were spotted as well:

- Sometimes, we failed to ensure that all participants had understood a topic well enough to participate further. This led to diverging groups, in which novel algorithms were developed by those still able to follow the train of thought. Simultaneously, some students remained on their own and did not take part in the discussions until the group met again. For future sessions, we decided to discuss, document and visualize new ideas more thoroughly.
- For some research ideas our students developed, next steps remained unclear and nobody was assigned to drive them forward. Essentially, lacks in overall project management accounted for ideas getting lost and rediscovered later on. We improved the project management tools used (Kanban boards with dedicated assignees) to overcome this issue.

The synchronization session was meant to update different focus groups with the results of the other students. Furthermore, intermediate presentations should

help account for the appropriate reflection upon the executed research tasks. As pointed out in [6,23], this helps to avoid focusing to heavily on execution without evaluating results and lessons learned. These sessions increase student's opportunity for self-assessment and revision, one of the principles of successful inquiry-based courses stated by Barron et. al. [4].

As stated by the lecture reviewers, student presentations were sluggish, most likely caused by their inappropriate preparation. In retrospect, we identified our imprecise and not explicitly given expectations as the most likely reason. We suspect that students did not get the overall concept of the presentation & synchronization lessons. Without realizing what we aimed at, they were unable to perform appropriately.

Again, we adapted following sessions by supplying a coarse outline when asking for presentations: presentation of technique used, blockers and intended solutions used and open issues that could then be solved by the whole group.

To increase commitment, presentations had to be discussed with the lecturers beforehand in order to ensure quality requirements were met. Overall, following sessions were able to distribute individual knowledge to the other participants as intended.

Student feedback For the inquiry-based course, we performed several intermediate online evaluations. We asked students for their workload, motivation and an individual estimate of their learning outcome. Each was to be rated on a scale from 1 (lowest) to 5 (highest). Furthermore, we asked students to give a reasoning for their ratings using free-text answers.

The overall feedback was very positive and encouraging. In particular, the switch from a lecture-based to an inquiry and research-based design was successful:

- The course was described as interesting,
- Student evaluated their individual learning outcome as high (average of 4.375), mostly attributed to the increased self-reliance.
- The course was described as very work intensive, with an average of 3.75. However, the overall hours dedicated to the course by students was in alignment with the ECTS awarded. Thus, we deem the workload appropriate.
- Even though workload was high and the course was described as quite demanding, overall motivation was rated with an average of 4.625.

The most dominant point of criticism was the volatile speed of progression. While it is certainly impossible to guarantee a progression speed while allowing individual research, the overall progress had to be ensured better and in a way that was obvious for the students. This criticism drove some changes in the later course iterations.

We were pleasantly surprised by the very highly rated motivation. However, individual motivation is hard to compare to other courses judging by the self-assessment alone.

Thus, we tried to empirically evaluate student motivation using the activity and commit data available for the repository[4] used to develop the model checker. To summarize [16]:

- Most changes were made during the sessions in presence.
- Consistent activity throughout weekdays and working hours.
- Higher activity before lectures, maybe due to students revising the material.
- Occasionally, we see a student working all night. In a second evaluation, students stated that this was not caused by an overboarding work load, but to individual interest, motivation and time management.

Another indication for high motivation due to our research-based approach is that three of the participants worked with us towards a publication of their research results. This was an optional offering, not linked to the course, its grading or rewarded with credit points. Still, students put in further effort and managed to publish a paper [20].

Reflection

Scalability and repeatability We do not deny that a group of students is able to write an interesting, somewhat sophisticated and useful program during a semester. However, due to the size and requirements of the programming project, different skills and programming experiences were needed for success. Yet, we cannot reasonably expect those skills to be available in following iterations of the course. Additionally, performing a joined programming project does not scale arbitrarily. This approach would not work with semesters where too few or too many students attend. Of course, one could create several group competing in, e.g., performance. Yet again, different skills should be present in each group and one would have to deal with assigning students to groups accordingly.

Grading Grading students based on their involvement in a programming project is hard and often not as objective as desired. Objective metrics, such as the number of commits or lines of code give no insight into the students' knowledge and understanding.

Knowledge propagation Students tend to acquire more knowledge about their own area of focus. Since the research work was split into different topics, knowledge often did not propagate equally. In our experience, this holds true for both seminars and programming projects.

Too rich formalism The B language is very expressive. While this allows for concise and precise specifications, evaluating state transitions is complicated and needs constraint solving algorithms, e.g., to compute parameters. Thus, even the subset of B given to the students required considerable work on implementing a language interpreter, effectively diverting resources from the actual model checking algorithms. Yet, this iteration raised a very important question that we shall return to later on: "*How formal should formal methods be taught?*"

[4] `https://github.com/bmoth-mc/bmoth`

3.3 Second Iteration: Lessons Learned: Mixing Lecture and Practical Exercises

Goals and Ideas Due to the concerns mentioned above, we decided not to repeat the course without further modification. Instead, we tried to combine the best of both worlds by teaching theoretical aspects using lectures while also making students work on individual, smaller-scale programming projects.

Teaching & Course Execution With this iteration, we started from the initial set of lectures again. We cut back on long proofs, reduced the build up to important theorems and provided smaller exercises that still kept the main idea.

Instead, students were supposed to individually implement a model checker with LTL capabilities. This time, the formalism was kept intentionally simple (i.e., petri nets), so a naive reachability tool can be implemented in a few hours.

We also provided parsers for the models and LTL formulas, as well as a transformation of formulas in positive normal form to generalized non-deterministic Büchi automata. Learning from the overhead of implementing a language interpreter, this allowed students to focus on the model checking aspects.

This programming project is sufficiently small that it could be done in an appropriate amount of time, yet also forces students to internalize the required steps for LTL model checking.

Grading Additionally to the summative exam at the end of the semester, we kept a formative part of the grade: A reduced version of the model checker project, that can be used for reachability analysis, deadlock and invariant checking, was mandatory to take an exam. Full LTL capabilities of the resulting tool made up 20 % of the overall grade.

Student Feedback Two points stand out in the evaluation: firstly, all students still attending the course rate it excellent in structure, materials, lecturer and overall impression. Furthermore, students rate their subjective learning success as excellent to good.

Additionally, the programming project was highlighted positively in an open question. However, compared to other courses, the initial interest is rated rather low: on a scale of 1 (best) to 5 (worst), the median 3 and arithmetic mean 2.4.

Reflection Overall, we think this course is an appropriate compromise. It ensures that students follow the correct path, yet enforces hands-on experience. However, grading proved to be difficult.

Grading When reviewing and testing the students' code it is usually all or nothing. Grading programming projects without clearly communicated criteria is not feasible. One has to decide, whether only to grade functionality and correctness, or to add criteria such as code style, performance, etc., and how to grade those.

3.4 Third Iteration: Improved Teaching Methods and Online Teaching

Goals and Ideas In 2020, due to the COVID-19 pandemic, we had to make a quick leap to online teaching with very limited preparation. Additionally, the course was held in 13 rather than 15 weeks, as the semester was shortened because of the pandemic. This did not allow us to continue the mode of the prior iterations and forced us to try something new. In particular, these conditions did not allow us to include programming work without abandoning important content.

The situation suggested an inverted classroom (see, e.g., [26]): Instead of giving a lecture and having the students prepare exercises at home, we met for weekly exercise sessions with reading tasks to be done individually. With online versions of standard books on model checking [3,9] and lecture recordings by the RWTH Aachen University[5], this was possible without long preparation.

Execution In this iteration, the course followed the first five chapters of PoMC with a final glimpse at timed automata. Note that this involves a large share of mathematical notation, proofs, etc. Course organization and philosophy were heavily inspired by Keller's Personalized System of Instruction [14]:

On-line sessions One goal was to limit the time spent interacting online and invest it into self-study instead. In particular, no lecture was held. This decision comes with the idea that the — mostly mathematical — load during a regular lecture is far too high for a student to actually follow and, thus, participate. In the past, certain proofs (e.g., correctness of the nested depth-first search) were guaranteed to outpace the majority of students. Instead, a typical session was structured as follows:

1. Opening and mood barometer: as part of the opening, it was important to us to poll the current mood of the students, and, to build trust, give our own. Usually, it was used to gain insights on key questions (e.g., the students' perception of the new course iteration, their current situation during the pandemic, their happiness with online courses overall). For this, a slide with a 4x4 grid of emojis was prepared that students were able to point to and draw at. Additionally, everyone could comment via voice or chat. This method allowed some initial activation and personal interaction, which — in our case — led to a good course atmosphere.
2. Material review and electronic voting system (EVS): Miller and Cutts described benefits of an EVS in a formal methods course [18]: In particular, they used it to ensure that students read the material and understood it, and students became more confident in their knowledge and were more willing to answer questions. Thus, in the next part, we prepared some single-choice questions on the material for revision and as a light means of testing understanding. Occasionally, questions were designed to trap students with

[5] https://www.youtube.com/playlist?list=PLnbFC0ntxiqdpoWwMKCVh6BRwBePHaqQx

fallacies that we have observed in the past[6]. Naturally, these were discussed more in-depth afterwards.
3. Live exercises: For the majority of the session, students were given the opportunity to ask questions on the material and to choose the exercises they want to solve (e.g., taken form the large number of questions in [3]). For the key concepts, we prepared questions that showed on a minimal example how a technique works. Proofs and answers were written co-operatively on a shared whiteboard, with the lecturer only adding notes, guiding the students if stuck or correcting errors.
4. Outlook and intuition: Again, the idea is that the most important benefit of a lecture is that students develop an intuition, yet have to work out details at home at their own pace. During the last minutes, we tried to give an intuition on the material that should be prepared for the next session. We only sketched connections to prior material and the basic idea, without formal definitions, proofs, etc. Instead, we raised key questions the students should find the answer to.

Learning Units The course was structured into units that were made available for the students entirely at the beginning of the semester. Each unit contained references to learning material (i.e., relevant sections of books, lecture recordings and scientific articles), a list of expected learning outcomes, a brief enumeration of the most important concepts of the unit, and a collection of exercises.

Learn at your own pace Students were given the opportunity to choose their own pacing which they deem suitable for their learning style. This includes both the individual speed (which — in a traditional setting — is dictated by the speed of the lecturer) and the time during the semester they learn (each week, or starting a few days before the exam).

To motivate continuous work and ensure students prepare for the online sessions, they were allowed to hand in a learning diary before the session. This learning diary may include anything related to the learning unit (though not simply copies of textbooks), e.g., notes on definitions or solutions to exercises they solved themselves, and was allowed as individual resource during the exam. A short version limited to two pages was allowed to be handed in up to a week afterwards, in case personal circumstances (sickness, etc.) rendered it impossible for some students to complete it in time. All learning diaries were inspected by a teaching assistant and errors were marked.

Feedback

Peer review For the online setting, peers and the didactics department were satisfied with the methodology. They also were able to identify issues related to eLearning. Their highlights include the following:

[6] Aiming at finding questions and distractors to eventually be used for peer instruction [10]

- The mood barometer is a nice method for activation, with the students and lecturer even revealing personal insights.
- The use of revision and EVS is good. Yet, during discussion of follow-up questions students often are unsure whether to use their microphone, chat or wait for voting options.
- The methodology of assigning the task of creating their own summaries (e.g., what are the steps required for LTL model checking) and take home-messages (e.g., how do counterexamples to different kinds of properties look like), to students is important to deepen their understanding.
- The students are very engaged and take initiative during the sessions in creating and discussing solutions.
- Nonetheless, student webcams remain deactivated.

Student feedback For the online course during 2020, we polled the students after the course. A four-point scale of strong/weak agree/disagree with an option of not applicable was used. Students unanimously (strongly, unless stated otherwise) agreed on the following:

- The structure of the course is excellent,
- the inverted classroom setting was worthwhile,
- the online session was very useful (one weak agreement),
- the PoMC book is very understandable (one weak agreement),
- the lecture recordings of Prof. Katoen are very understandable,
- explicitly-stated learning outcomes helped their self-study,
- overall, they are very satisfied with the course.

On the following two statements, one student weakly disagreed while all others agreed: "An inverted classroom would be worthwhile in an off-line setting", and "The outlook part of the session was helpful".

In an open question, a student highlighted that they felt that nobody was left behind in case there were questions of uncertainties and that everything was discussed until everyone understood the matter.

Reflection

On establishing a testing culture Due to a small course size, the learning diaries and a good atmosphere during the online sessions, it was possible to ensure that all learning outcomes were met before the exam, both for the students and the lecturer.

In retrospect, we would recommend online testing of students, where students have to achieve very high marks for every unit, yet may attempt a test as often as necessary. In another course, it has proven valuable to discuss every test briefly (i.e., 5 – 10 minutes) with the individual student, probing for understanding of the matter and correcting minor mistakes. Teaching assistants can share this load with a lecturer or even do the work entirely.

Table 1. Average Grades (in Parentheses: Adjusted Data Without Failing Students / Without Formative Grades)

Year	# enrollments	# exams	∅ grade	course type	exam modus
2014	11	2	1.85	lecture-based	written
2015	12	(4) 5 (1.98)	2.58	lecture-based	oral
2016	13	7	1.71	lecture-based	written
2017	11	6 (1.43)	1.28	inquiry-based	written + formative
2018	18	5	1.88	lecture + programming	written
2019	10	5	1.58	lecture + programming	written
2020	16	5	1.54	online + inverted classroom	written

Exemplary solutions An opinionated topic is whether, when and how solutions to exercises should be made available to students (e.g., [19]). While we cannot give an ultimate answer to these questions, we noticed that, often, exercises requiring proofs (e.g., classification of properties as safety or liveness properties) contained errors — even though a solid mathematical education is required to attend our course. To counteract, we think that in these cases an annotated and correct solutions should be made available to the students, even if it is to simply ensure that they have seen a correct solution and what fallacies need to be avoided.

No hands-on activity Even though no programming activity was mandatory during the semester, students were aware that they might face programming tasks in the exam as stated in the expected learning outcomes. In fact, when provided with a small interface, all students were able to implement a variant of the model checker, that was required for admission the years before, without errors.

4 Comparison of Grades

One of the measurements — and sometimes the only — of learning success are the grades given at the end of the semester. In this section, we give an overview of the grades throughout the years and discuss how they can be compared.

Grades are given on a scale from 1 (excellent) to 5 (fail) in steps of 0.3. An overview of the last years is given in Table 1. Note that in 2015, a failing student may distort the data, and the adjusted value is given as well. In 2017, when including the formative part of the course, i.e., implementation work and participation, the average grade improves as well. The value of the exam alone is additionally given in parentheses.

In general, it is hard to draw reliable conclusions from relatively small sample sizes of five to seven exams per year. However, one can identify certain trends that align with different teaching methodology:

When considering the data, one can see that students attending the inquiry-based course in 2017 were more successful than those attending the purely

lecture-based courses since 2014. Especially with the formative part, the average grades improve significantly. One explanation might be that the practical research experience deepened the students' understanding of the course matter much better than attending lectures. On the other hand, the student feedback also discloses a much higher workload than their other courses.

Students attending the second iteration in 2018 were on par with the lecture-based courses and improved in 2019. A possible reason for this might be the additional experience gained with this teaching style and, thus, better supervision of student activities. In the online course from 2020, students performed slightly better, and the amount of excellent grades increased as well.

One interesting outlier is the course from 2015, where oral exams were held rather than written ones. An explanation for the significant lower grades might be that students were not used to talk about course matter, and thus achieved lower grades in that setting.

5 Conclusions

In this paper, we presented several iterations of our course on model checking, describing a shift from lecture-based teaching to more self-responsible learning, increasing participation and practical experiences. With these experiences, we will return to the issues raised in Section 1 and present our conclusions.

How formal should formal methods be taught? We think that student engagement in lectures heavily benefits from being more informal, i.e., avoiding strictly mathematical discussions and long proofs. All hands-on exercises, may it be programming tasks or exercise questions that engage students in discussion, are preferable. The formal aspects of formal methods can be taught as part of reading tasks instead, as we have done in the online course. Additionally, we argue that the formal parts of formal methods should be taught only after the benefits of using formal methods are understood by the students.

To what extent do students benefit from practical experience? As reported, student feedback on practical projects has always been positive.

Practical experience in implementing presented algorithms provides a second gateway to the desired learning outcomes[7]. We suspect that after implementation students have more in-depth knowledge, have realized certain edge cases and are more likely to remember technical details.

As an alternative to implementing model checkers from scratch, examining existing tools such as ProB could be considered. This however comes with an entry barrier. For instance, ProB is implemented in Prolog which students usually do not know well. Furthermore, the code base is large and code snippets might need a lot of context to be understood.

[7] Keep in mind that modelling systems and applying model checkers to them is taught in a seperate course.

Also, modern model checkers rely on many other techniques students did not implement, e.g., BDDs or partial order reduction. In summary, existing implementations are often more confusing than helpful and clean room implementations are to be preferred.

Overall, the benefit of having students implement a model checker may be lower than we hoped for when considering potential for student theses. Nonetheless, it seems to be a valuable addition to a course on model checking.

How to increase student interest? How to improve student perception of formal methods? This question cannot be answered conclusively from our experience. Simply put, it is too late to reach students once they decide to choose another course over a formal methods one. One possibility is to tighter integrated formal methods with other courses in the computer science curriculum (as suggested by [8]). An alternative could be to advertise FM courses with innovative and fun teaching concepts.

However, once they reach the classroom, we believe that with properly designed courses we can convince students that formal methods are more than boring mathematics, and tools are not black magic.

There are many suggestions for courses focusing on modeling and proof, e.g., using games [15] or puzzles [24] as examples. Judging from student feedback, we think that programming tasks or letting students find answers to certain questions themselves are an engaging and, ultimately, fun approach.

Inverted classrooms and online sessions We argue that, especially for online teaching, an inverted classroom is a viable alternative — especially, when a course follows a more formal approach. In a lecture-based approach, there is rarely time to comprehend new mathematical formulas and proof during lectures. Yet, even in a less formal format, students can heavily benefit from acquiring theoretical knowledge at home and use the time of synchronous sessions for discussions instead.

Acknowledgement. The authors would like to thank their peer lecturers Jens Bendisposto, Natalie Böddicker, Janine Golov, Ann-Christin Uhl and Susanne Wilhelm for reviewing their courses. They also thank Joshua Schmidt for his input, fruitful discussions and course execution in 2019.

References

1. Abrial, J.R.: The B-book: Assigning Programs to Meanings (1996)
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
4. Barron, B.J., Schwartz, D.L., Vye, N.J., Moore, A., Petrosino, A., Zech, L., Bransford, J.D.: Doing With Understanding: Lessons From Research on Problem- and Project-Based Learning. Journal of the Learning Sciences **7**(3-4), 271–311 (1998)

5. Biggs, J.: Enhancing teaching through constructive alignment. Higher education **32**(3), 347–364 (1996)
6. Blumenfeld, P.C., Soloway, E., Marx, R.W., Krajcik, J.S., Guzdial, M., Palincsar, A.: Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. Educational Psychologist **26**(3-4), 369–398 (1991)
7. Butler, M.J., Hallerstede, S.: The Rodin Formal Modelling Tool 1. BCS-FACS Christmas Meeting (2007)
8. Cerone, A., Roggenbach, M., Davenport, J., Denner, C., Farrell, M., Haveraaen, M., Moller, F., Körner, P., Krings, S., Olveczky, P., Schlingloff, B.H., Shilov, N., Zhumagambetov, R.: Rooting formal methods within higher education curricula for computer science and software engineering – a white paper. CCIS, vol. 1301. Springer (2021), `https://arxiv.org/abs/2010.05708`
9. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
10. Crouch, C.H., Mazur, E.: Peer instruction: Ten years of experience and results. American journal of physics **69**(9), 970–977 (2001)
11. Geleßus, D., Leuschel, M.: ProB and Jupyter for logic, set theory, theoretical computer science and formal methods. In: Proceedings ABZ 2020. LNCS, vol. 12071, pp. 248–254. Springer (2020)
12. Grumberg, O., Veith, H. (eds.): 25 Years of Model Checking: History, Achievements, Perspectives, LNCS, vol. 5000. Springer (2008)
13. Holzmann, G.J.: The model checker spin. IEEE Transactions on software engineering **23**(5), 279–295 (1997)
14. Keller, F.S.: Good-bye, teacher.... Journal of applied behavior analysis **1**(1), 79 (1968)
15. Krings, S., Körner, P.: Prototyping games using formal methods. In: Proceedings FMFun 2019. CCIS, vol. 1301. Springer (2021)
16. Krings, S., Körner, P., Schmidt, J.: Experience report on an inquiry-based course on model checking. In: Proceedings SEUH 2019. vol. 2358, pp. 87–98. CEUR (2019)
17. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. STTT **10**(2), 185–203 (2008)
18. Miller, A., Cutts, Q.: The use of an electronic voting system in a formal methods course. In: Proceedings FM-Ed 2006. pp. 3–8 (2006)
19. Nygren, H., Leinonen, J., Hellas, A.: Non-restricted access to model solutions: A good idea? In: Proceedings ITiCSE 2019. pp. 44–50. ACM (2019)
20. Petrasch, J., Oepen, J.H., Krings, S., Gericke, M.: Writing a Model Checker in 80 Days: Reusable Libraries and Custom Implementation. ECEASST (2018)
21. Radermacher, A., Walia, G.: Gaps Between Industry Expectations and the Abilities of Graduates. In: Proceeding SIGCSE 2013. pp. 525–530. ACM (2013)
22. Radermacher, A., Walia, G., Knudson, D.: Investigating the Skill Gap Between Graduating Students and Industry Expectations. In: Proceedings ICSE 2014. pp. 291–300. ACM (2014)
23. Schauble, L., Glaser, R., Duschl, R.A., Schulze, S., John, J.: Students' Understanding of the Objectives and Procedures of Experimentation in the Science Classroom. The Journal of the Learning Sciences **4**(2), 131–166 (1995)
24. Schlingloff, B.H.: Teaching model checking via games and puzzles. In: Proceedings FMFun 2019. CCIS, vol. 1301. Springer (2021)
25. Tafliovich, A., Petersen, A., Campbell, J.: On the Evaluation of Student Team Software Development Projects. In: Proceedings SIGCSE 2015. pp. 494–499. ACM (2015)
26. Talbert, R.: Inverted classroom. Colleagues **9**(1), 7 (2012)