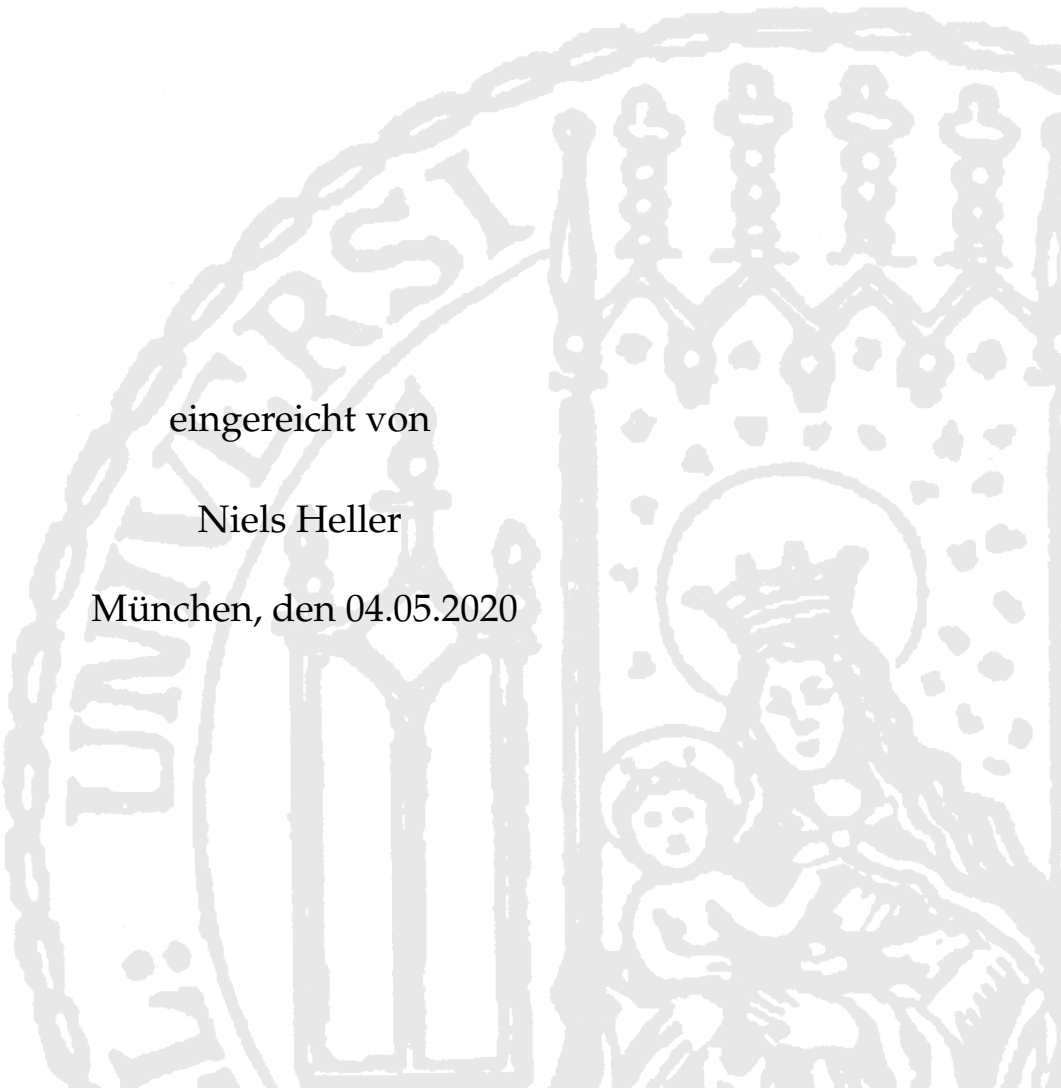
Pervasive Learning Analytics for Fostering Learners' Self-regulation

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München



eingereicht von

Niels Heller

München, den 04.05.2020

1. Gutachter: Prof. Dr. François Bry

2. Gutachter: Prof. Dr. Carlos Delgado Kloos

Tag der mündlichen Prüfung: 21.07.2020

Eidesstattliche Versicherung
(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, den 05.05.2020 Niels Heller

Abstract

Today's tertiary STEM (Science, Technology, Engineering and Mathematics) education in Europe poses problems to both teachers and students. With growing enrolment numbers, and numbers of teaching staff that are outmatched by this growth, student-teacher contact becomes more and more difficult to provide. Therefore, students are required to quickly adopt self-regulated and autonomous learning styles when entering European universities. Furthermore, teachers are required to divide their attention between large numbers of students. As a consequence, classical teaching formats of STEM education which often encompass experimentation or active exploration, become harder to implement.

Educational software holds the promise of easing these problems, or, if not fully solving, at least of making them less acute: Learning Analytics generated by such software can foster self-regulation by providing students with both formative feedback and assessments. Educational software, in form of collaborative social media, makes it easier for teachers to collaborate, allows to reduce their workload and enables learning and teaching formats otherwise infeasible in large classes.

The contribution of this thesis is threefold: Firstly, it reports on a social medium for tertiary STEM education called "Backstage 2 / Projects" aimed specifically at these points: Improving learners' self-regulation by providing pervasive Learning Analytics, fostering teacher collaboration so as to reduce their workload, and providing means to deploy a variety of classical and novel learning and teaching formats in large classes. Secondly, it reports on several case studies conducted with that medium which point at the effectiveness of the medium and its provided Learning Analytics to increase learners' self-regulation, reduce teachers' workload, and improve how students learn. Thirdly, this thesis reports on findings from Learning Analytics which could be used in the future in designing further teaching and learning formats or case studies, yielding a rich perspective for future research and indications for improving tertiary STEM education.

Acknowledgments

First and foremost, I would like to thank Professor Francois Bry for his help and constant support and for his most valuable advice on all kinds of matters.

Secondly, I would like to thank Sebastian Mader for setting up Backstage 2 with me, attacking the thankless task of sustaining the document service, his constant and enthusiastic help, and especially for his company in the conferences we attended together.

Thirdly, I would like to thank Elisabeth Lempa for sticking around in the cellar of our institute, "executing" the most valuable proof reading, developing CoCoNut with all her imagination and creativity, and especially for her ability to calmly and cleverly point out different perspectives.

Special thanks go to: Mathias Schlenker for his work on the representation of systematic errors and for polishing code (which was totally unasked and very welcome!), Andreas Born for his work on predicting systematic errors and on improving the prediction unit, and to Steven Dostert for his work on predicting skipping and devising the prediction unit.

Furthermore, I would like to thank, Zorig Dorj, Joana Haag, Marco Hoffman, Florian Holzinger, Galina Keil, Caroline Marot, Robert Pospisil, Julian Reff and David Tellenbach for their contributions to the software.

Last but certainly not least, I would like to thank my wife Dörte for always being there, supporting me in all stages of this doctoral project, and for contributing more than once her ideas to the research project.

Contents

CHAPTER 1

Introduction

This report presents the conception, implementation, and evaluation of the educational software "Backstage 2 / Projects" and its ecosystem which has been designed to address problems inherent to tertiary mass education in STEM (Science, Technology, Engineering, and Mathematics). The following paragraphs introduce this work by briefly discussing the relevant terms (namely: Tertiary mass education, self-regulated learning, STEM education, and Technology Enhanced Learning), summarizing the results of the presented research, and laying out its structure.

Mass education. The term "mass education" is commonly used to describe education systems which accommodate large parts of a country's population [24]. Europe's universities have become institutions of mass education over the last century [188, 179], a process often attributed to the economic advantages of graduation and the democratization of education in the second half of the 20th century [114, 188]. This process of university "massification" [114] has proceeded in the last decades: In their extensive report on academic staff in higher education and their recent challenges in Europe, Crosier et al. [53, Chap. 1.1.1] found an increase in student enrolment numbers in tertiary education programmes of 40% between the years 2000 and 2015. Crosier et al. point out that these numbers have to be interpreted with caution, due to, among other reasons, the difference in population sizes and demographics between European states, and incomplete data. STEM fields, which constituted the second-largest fields of study by student numbers in Europe in 2018 [82], are no exception to this development. The number of STEM students grew with a rate comparable to the growth rate across all fields: 37% between 2003 and 2012 [4].

Tertiary mass education is often associated with large class teaching [114, 10], with course sizes of several hundred students being common in today's European universities. The resulting very large student-teacher ratios, which are known to impede student success [166, 139, 169, 18], are a persistent problem in Europe: While Crosier et al. found that academic staff sizes grew during the last decades, they conclude that the numbers of academic staff did not grow consistently with student numbers [53]. Indeed, comparing both growth rates between the years 2000 and 2015 yields that student numbers grew in

average 3.9 times faster than staff sizes.[1] It has to be noted that these numbers are national averages, and the situation might differ between universities or even faculties. As an indication, the student-professor ratio grew from 142 students per professor in 2014 to 212 students per professor in 2018 at the author's department of Mathematics, Informatics and Statistics.

Tertiary mass education and large classes entail a multitude of problems, three of which seem particularly striking:

1. Large classes foster **passivity** among students, who show less engagement and lower motivation than students taught in small classes [169, 253].
2. Large classes severely **limit teacher-student contact and interaction** [81, 169] which limits the amount of feedback teachers can provide to students. Note that both feedback provided by teachers to students [102] and provided by students to teachers [101, 171] is considered largely beneficial for student learning.
3. Large classes foster **social isolation** and an impression of anonymity among students [253, 22, 243] which can in turn "reduce students' sense of responsibility for class interaction" [114].

Self-regulated learning. A general approach to solving problems caused by large classes is to foster self-regulated learning. Self-regulated learning theory describes how students "personally activate, alter, and sustain their learning practices in specific contexts" [267, p. 307]. Several authors identify different components of self-regulated learning, among others: Goal orientation and motivation [189, 190], meta-cognition [190] (in the sense of reflection on one's own learning process), help-seeking [154, 31], and learning organization [154, 31] (such as time management, choice of a learning method, etc.). Undoubtedly, learners endowed with such skills would be less subject to the aforementioned problem of passivity and inactivity, while being better at coping with the problem of limited teacher contact through better help-seeking abilities.

Tertiary STEM education. The goal of this research is to improve tertiary STEM education, yet the term STEM has different connotations depending on the perspective (educational or political) it is used for. The term was coined by the US American National Science Foundation in the late 1990s [218] and was since then often used by western governments to label educational funding programs which pursue vocational and economic goals [23, 258]. From the educational perspective, the term is often criticized: In secondary education there usually are no engineering courses [23], there is no consensus on what is exactly included in the technologies the "T" stands for [258], and in tertiary education the term just refers to a set of separate fields which are often taught independently of each other, hence making the introduction of a combining term inappropriate [218]. In the late 2000s, the term "integrated STEM education" emerged [23] which refers to methods of integrating the content of several STEM disciplines into one course or lesson, for instance by using an engineering problem to introduce a new mathematical principle [167]. Integrated STEM approaches often rely on active teaching methods such as problem-solving and inquiry learning [23].

While interdisciplinary STEM teaching is arguably an interesting approach to improve motivation and learning in these fields, this work is devoted to a different issue: It tries to identify teaching and learning problems in tertiary STEM education as it currently is, and to solve (or to alleviate) these problems by deploying educational software.

Technology Enhanced Learning. This report is a contribution to the research field of Technology Enhanced Learning (TEL), which can be defined as the "application of information

[1] Based on Eurostat tables `educ_enrl1tl`, `educ_uoe_enrt02`, `educ_enrl1at`, `educ_uoe_enrt01` used in [53], by comparing only countries where complete datasets were available.

and communication technologies to teaching and learning" [133]. Kirkwood found in a literature review that most of the examined TEL applications were either used to replicate or supplement existing teaching practices (such as providing video recordings of lectures), with only about a third of the applications using technology to restructure or change the learning and teaching process [133]. Similarly, Henderson et al. found recently that university students used technology mostly for organizational purposes, such as locating and retrieving lecture content and handing in homework assignments [111].

In this way, TEL applications are already widely used to solve *organizational* problems of mass teaching: It is nowadays unthinkable to organize for instance homework assignments or to provide lecture material for hundreds of students without software. The general contribution of this report, however, is to introduce and evaluate educational software designed to solve the specific problems of tertiary mass STEM *learning*. To pursue this goal the software described in this report provides Learning Analytics that are adjusted to tertiary STEM education.

Learning Analytics. Learning Analytics use techniques from Statistics and Machine Learning to accomplish educational goals, and can be defined as "the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs" [228]. The most common approaches to improve learning through learning analytics are presenting descriptive statistics (such as summaries of recorded learner data), alerting students in case of detected problems (such as high risks of failing a course), and adapting or choosing course material according to the students' needs (such as choosing exercises of a suitable difficulty) [181, 85].

Contribution of this report. The contributions of this report are as follows:

- A web-based educational software is introduced, its implementation, the software eco-system it is embedded in, and several learning and teaching formats supported by the software are described. The software aims to alleviate problems occurring in tertiary STEM education, and design decisions which reflect the special requirements of these fields of study are highlighted.
- Learning analytics which are provided by the software, as well as their pedagogical and scientific use, are described. The pedagogical use describes how these analytics can be (and were) used in a course to solve an educational problem (such as reducing passivity or absenteeism); the scientific use is to provide insights in learning behaviours of students in the case studies and to examine the efficacy of the software for learning.
- The results of several evaluations, gathered either in case or laboratory studies, are reported: Collaboration on the platform, both of learners and teachers, helped learning and teaching. Behavioural changes indicating higher self-regulation could (with some limitations) be triggered by the software using Learning Analytics. Exploratory learning was fostered (students used the software for voluntary elaboration and experimentation), and the attitudes towards the software were generally good among students and teachers alike.

Several case and laboratory studies were conducted for this research, each evaluating and experimenting certain of the software's aspects. As many of the results were published (either in articles or in student theses), this report aims to summarize general results and implications that can be derived from the research as a whole, therefore the same case study may be referenced in different sections. A list of the author's publications cited in this thesis, naming the main contributors to these publications, can be found in Appendix D. All case studies were conducted in courses held at the author's department of computer science, and a list of all these courses as well as short descriptions of the regarding topics can be found in

Appendix A.

This report is organized as follows: The next chapter describes the specific problems of tertiary mass STEM education and motivates choices in the design of the software. Chapter 3 describes related work by discussing related educational software, Learning Analytics approaches, and several learning theories. Chapter 4 describes the implementation of the software by first describing its components and then introducing several teaching formats that can be supported by these components. Results on the students' and teachers' attitudes towards these formats, as well as general observations can also be found in this chapter. Chapter 5 discusses Learning Analytics integrated into the software, which aim to predict learner behaviour. Chapter 6 presents and discusses results regarding learning and teaching effectiveness. Chapter 7 discusses perspectives for future research and implications for the design of tertiary mass education in STEM fields, and Chapter 8 concludes this report.

CHAPTER 2

Motivation

This chapter motivates both software and research design choices by analyzing problems specific to STEM mass education. Further, limitations of the scope and the methodology of this research are noted.

To achieve the aforementioned goals (providing and evaluating software solutions to the problems specific to tertiary mass STEM education), particularities in the ways STEM subjects are *learned* have to be considered. While there is no "STEM learning" theory in general, three observations which seem relevant for software design can be drawn from the literature:

1. **Active Learning** approaches seem particularly suited for STEM education. Active learning is defined by Prince as "any instructional method that engages students in the learning process" [196, p. 1] and is often contrasted with "passively listening to a lecture" [196, 7]. While the plethora of teaching methods encompassed under the term "Active Learning" may apply to other fields, the term is commonly used in the STEM education literature (see for instance [90, 151, 237] for meta-analyses of Active Learning methods which solely rely on evaluations in STEM fields).
2. Learners in STEM fields often make **systematic errors** or have **systematic misconceptions** while learning. Notable properties of these misconceptions are their consistency across ages and abilities, and their "resistance to change through traditional instruction" [45, p. 4]. Teaching in STEM education often encompasses teaching to overcome misconceptions, and methods which account for this fact are often inspired by the "conceptual change model". It describes how misconceptions are exchanged for other (hopefully) better conceptions [192]. While the terms misconception and conceptual change are traditionally used in the science education literature, it has recently been used to describe learning problems in computer science [198] and mathematics [201].
3. STEM fields rely on **formal languages** for their discourses: Programming languages in computer science, structural formulas in chemistry, differential equations in calculus, and so on. These formal languages are (in contrast to natural languages) non-ambiguous and dense in the sense that they convey a lot of abstract information with few symbols, and are often perceived as counterintuitive and challenging by

learners [216]. To use formal languages, STEM learners must learn to master their syntax, and numerous studies report on the problems of students in facing this task: See for instance [96, 122, 148, 150] as of computer science and [134, 25, 200] as of secondary mathematics education.

It has to be noted that these observations are neither exclusive to STEM education, nor can they be regarded as a complete description of STEM learning. For instance, peer instruction and peer review, which certainly are Active Learning methods, are widely used in non-STEM fields [241], and systematic errors might very well occur while, for example, learning a secondary natural language. Also, there surely are further important aspects of STEM learning which are not addressed in this report. Yet, it is evident that these observations conflict directly with the problems of mass education: While in the literature, Active Learning approaches are praised especially for STEM fields, the current state of affairs in European tertiary education seems to encourage (or at least foster) passive student behaviour. And while Active Learning methods are attempted in large classes, they seem to be most effective in class sizes below 50 students [90]. The literature, especially in the tradition of social constructivism, praises direct interaction and tutoring for helping students building knowledge [252, 126], and teaching methods that facilitate conceptual change often encompass active experimentation and classroom discussion [224] – which conflicts with the widespread shortage of teacher-student contact in European tertiary education.

As a result of this analysis, the software introduced in this report pursues three goals: Supporting and encouraging active and self-regulated learning, identifying and helping students in overcoming systematic errors and misconceptions, and finally helping students in using STEM's formal languages. These goals have to be pursued while requiring limited teacher involvement, or increasing teaching effectiveness – teachers are the limiting resource in mass education.

A key technique in this pursuit are Learning Analytics because they allow to "mimick" teacher behaviours typical to small class teaching: With Learning Analytics struggling students, as well as common problems, can be identified, and the learning progress of large groups can be automatically assessed. Also, with Learning Analytics, decision-making can be automatized: Choosing suitable partners for group work, providing personalized scaffolds, and nudging students who are dragging behind towards more participation. Hattie writes on the profession of teaching: "When these professionals (teachers) see learning occurring or not occurring, they intervene in calculated and meaningful ways to alter the direction of learning to attain various shared, specific, and challenging goals" [101, p. 22]. The notion of "calculated" interventions is interesting: Learning Analytics provide such calculations without relying on the teacher as a "calculator".

Interestingly, for the implementation of Learning Analytics, mass education and large class sizes are advantageous. Here, many students are taught under very similar conditions which are often reproduced with few changes in the following course venues. These conditions allow mathematical models (such as statistical procedures) to deliver more reliable results than they would in small classes. This means that, firstly, finding Learning Analytics approaches is easier in large classes (for instance because correlations that would be indistinguishable from noise in smaller datasets can be identified), and secondly that the effectiveness of introduced interventions can be measured more reliably.

The following paragraphs describe how the software was used in the teaching practice and how data collection was realized, what kinds of evaluations were conducted, and the limitations inherent to these approaches.

Software functionalities and use. The software was used to support several courses taken from the standard bachelor degree computer of science curriculum which were held at the

author's university. Hereby, the software provided functionalities supporting asynchronous learning activities, which do not depend on teachers being present and do not require an exact time of activity (consider in contrast attending a lecture held by a teacher on a specific date). Note that these activities are often self-regulated: Students *choose* to work (or not to work) on homework at a specific time or to post a question in an online forum. Therefore, examining the use of these functionalities seem appropriate for examining self-regulated learning. The software is a web application, which is the common technique for providing asynchronous learning services [205].

The functionalities provided by the software can be categorized as follows:

- **Learning management** functionalities: Organizing learning materials and their discussion, scheduling and assigning deadlines for homework and so on.
- Functionalities **imitating teacher behaviours**: These functionalities encompass identifying and nudging students towards better learning with predictive Learning Analytics (see Section 4.3), providing scaffolds and feedback while students work on a problem (see Section 4.5), and functionalities to intelligently orchestrate interactions between students (such as assigning suitable partners for peer review). While functionalities of the latter kind were implemented (and are operational!), evaluations of these are largely missing. Section 7.2 discusses perspectives for the use of these functionalities.
- Functionalities providing means for **collaborative learning and teaching**. As an example, the software allows teachers to collaborate while providing feedback on homework submissions (an evaluation of this functionality is found in 4.2), and for students to communicate and collaborate on their learning tasks (evaluations of these functionalities are presented in Section 6.2).

As data collection is both needed to fuel Learning Analytics during the courses and to evaluate the effectiveness of implemented measures, it was a major concern for the implementation of the platform. During a course, the software collects data both explicitly and implicitly. Explicit data collection refers to data that was directly entered by the users, for instance by teachers entering homework assessments, while implicit data collection happened without explicitly informing the users, for instance by logging the actions performed by students on the platform. After a course ends, the software stores the collected data as training data for Learning Analytics in future courses (see Section 5 for more details).

Evaluation approaches. The majority of the data evaluated for this research has been gathered in case studies, more precisely, by collecting data in courses that are part of every bachelor student's course of study at the author's faculty. While the data collected in this way may be confounded by non-controlled variables, this approach gives insights on the practical use of the software: After all, the goal is to solve hands-on problems in tertiary education. Also, in this way, data could be collected over long periods (courses take weeks or months), which allowed studying the change of behaviour during a course. Such time data series can hardly be obtained in laboratory studies.

In several instances, the repeating cycle of the bachelor degree course venues was used to examine the effectiveness of interventions in quasi-cohort studies by comparing data of consecutive course venues, where one was taught without and the next with an intervention (e.g. an additional software functionality) in place. While it might be more suited to evaluate the effectiveness of an intervention by applying it to a random sample of students of *one* course (forming the experimental group) and teaching the rest of the students as usual, such a method was not incorporated into the teaching practice: Firstly, this design would rely upon the ignorance of the subjects of their treatment (which could not possibly be upheld for hundreds of students over several months), and secondly, ethical problems arise as the evaluated interventions were supposed to improve learning. It is hardly ethical

to deprive a group of students of functionalities deemed helpful while supporting others with it. On the other hand, it is a common practice of teachers to evaluate new teaching methods for whole courses and compare the results to improve their teaching. One software improvement (aimed to scaffold students while learning simple programming languages) was not evaluated in a course, but in a laboratory study unrelated to the current courses (see Section 6.1).

"Learning effectiveness" is often measured by "examination success" in the following chapters. Note that this allows comparing examination results between students of the same venue (e.g. allowing statements like "students that used the platform in a certain manner were more successful in the examination"), yet *not* to compare examination results between venues: Examinations are not standardized and may yield different results because of differences in the examination questions. To test the efficiency of interventions, other measures such as comparing the error rates for specific exercises between course venues were used.

Limitations. While thoroughly chosen, several limitations of this research design have to be noted. Firstly, the evaluation is based on a very specific group of learners as only bachelor degree courses and topics were chosen for evaluation. These have the "advantage" of providing large numbers of students, which often have limited previous knowledge about the topics taught, yet it might limit the generalizability of the findings. Similarly, the evaluations all stem from computer science and mathematics courses, with no other STEM fields involved which might limit the generalizability of the findings to other STEM fields. Secondly, the assumption of "reproduced study conditions between course venues" stated earlier is flawed. Students enrol in several courses in parallel, and while the setting of a course may be quite accurately reproduced in the next venue, the setting of others may change: In one year a course might be especially well taught or the teachers may have decided to provide bonus points for participation – which may divert activity between courses. Indications for the occurrence of such problems are reported in Section 4.3. Also, parts of the teaching staff, exercises, and course content may change between course venues, which might limit the comparability.

CHAPTER 3

Related Work

This chapter reviews the literature on related work. The field of Technology-Enhanced Learning is both an applied field, as today's teaching and learning *practice* is usually supported by a variety of software, and a field of *theoretical research* which is rooted in learning theories, where new software is often being tested in laboratory settings. This chapter tries to account for this dichotomy of the teaching practice on the one hand and the theoretical research on the other.

Section 3.1 reports on the current use of software in tertiary mass education, by focussing on the use of social media and learning analytics. Section 3.2 discusses concrete learning software applications (which are often evaluated in laboratory studies). Here, a special focus lies on software fostering Self-Regulated Learning, sustaining collaboration and applications conveying STEM's formal languages. Section 3.3 discusses learning theories, which have informed certain design choices of Backstage 2 / Projects.

3.1 Software for Mass Education

Mass education with its large classes is a fact of tertiary education in many countries [114], and it is uncontested that increased class sizes have a detrimental effect on learning [231]. Using software, especially web-applications, to remedy these effects has been advocated for decades (see for instance [48] published in 2000). This section reviews reports on the current use and the reported benefits of educational software in tertiary education.

Software usage in tertiary education. In 2008, Dillenbourg wrote that the use of technology in education would become more and more natural, and would eventually blend into the practice just like other media have [64]. Indeed, Learning Management Systems (LMSs) are ubiquitous in today's universities. In a survey published in 2014, Dahlstrom et al. found that 99% of 151 evaluated universities in the US provided an LMS to its members [56]. Yet, while such systems provided a variety of features, most teachers and students only used the "basic" functionalities (allowing content management), with "advanced" features (allowing student collaboration) often being underused [56]. This is unfortunate because, as might be expected,

using software merely as a different medium to deliver the same content does not increase learning [133]. Chow et al. found in a comparative study that specific teacher training can diversify the use of educational software [44]. Hence, using interactive and collaborative features of an LMS seems to be a matter of training or habituation of the teacher.

Several authors argue for a less exaggerated language in the TEL literature, as the real-world applications of educational software seem to have much less revolutionary consequences for learning than many research articles suggest [64, 226, 111]. In general, the most beneficial aspects of software use in tertiary education seem to relate to content organization [111] and flexible access to learning materials [160].

Specific *software applications* however have been shown to improve learning in practice. Online homework systems for instance, which are often used in tertiary STEM education, can improve learning in mathematics [34], chemistry [208], and physics [61]. Often, these systems are equipped with automated grading or feedback mechanisms, which both reduces the teachers' workload and the time between homework-delivery and received feedback [197, 128]. An example of a technology-enhanced teaching method which can increase learning is the "flipped classroom", in which study material is provided online *before class*, and applications of the new concepts are acted out *in class* [13].

Social media in tertiary education. The term social media encompasses a large variety of web-applications with common functionalities being the integration of user-generated content, user-created personal profiles, and interactions between users (such as discussions) which allow the construction of social networks [174].

Using social media in education has been advocated among other reasons because they can foster collaborative learning (by allowing students to create and share content) [145], because they allow students to define their own (collaborative) learning spaces [55], and because it allows students to share their accomplishments with a wider audience outside the classroom (which can be encouraging) [142].

Yet, just as teachers have to be willing (and able) to use the collaborative features of an LMS, students have to be open to such uses of social media. To examine usage patterns of social media, White proposes the "visitors and residents model". Here, "visitors" use social media as a *tool* to achieve a specific goal (such as gathering information) while usually leaving few traces (generating less content). "Residents", in contrast, perceive social media as a social *space*, use it for social interaction, and usually leave more traces [257]. For collaborative learning with social media, "resident"-type behaviour would be preferable because students would be more comfortable discussing their learning progress with others [262]. However, students do not always use social offers for education in such a way: Druce and Howden found that a minority of examined students could be regarded as "residents" [72], similarly, Rambe found that most social media use for education was administrative and formal [202], and Ophus and Abitt found that students expected from an educational social medium mostly to "view their schedule" and "access notes and materials", and only rarely to use it for social interactions [177]. Junco found that performing social activities on Facebook like commenting was correlated to overall student engagement and co-curricular activities [123] which might suggest that "resident"-like behaviours are usually performed by students who are overall more social.

Teacher activities in social media are important. Wright argues that students will not show collaborative learning behaviours online if the teachers do not engage in such behaviours, too [262]. In a survey conducted with 333 university teachers, participants indicated that content sharing (between students and between students and teachers) was the most common use of social media, while collaborative components (opening a forum to discuss course content) being used less often [97]. In a study in 2012, Sadaf et al. found that about half of the

examined K-12 teachers were committed to integrating social media into their teaching while being uncertain how to do so [217].

Online Learning. Online Learning can be defined as "learning that takes place partially or entirely over the Internet" [163, p. 9]. In this definition, Means et al. subsume "blended learning" which is commonly defined as learning which involves both asynchronous online and synchronous face-to-face activities. In this sense, online learning is the most common form of learning in tertiary education, as Learning Management Systems are ubiquitous in today's universities [56].

Means et al. found in a meta-study that online learning per se can be advantageous over "face-to-face only" learning, among other reasons because it allows students to spend more time on specific tasks [163]. Yet, they also note that most reported advantages of online over face-to-face learning were likely to be caused by a change in pedagogy and instructional approach, not by a change in medium alone. For instance, they found that courses which were specifically designed for blended learning were more efficient than face-to-face or online-only courses.

A major concern expressed in the online learning literature is the social isolation which online environments can create [129], and the increased requirement of self-regulation and self-motivation [129], which are both capabilities not all university students possess [247].

Learning Analytics. A commonly used definition of Learning Analytics is "the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs" [228].

Three main approaches used to improve learning with learning analytics can found in the literature: Firstly providing descriptive statistics to the learners or teachers (such as summarizing reports on the previous learning behaviour), secondly alerting "at-risk" students (with high estimated risks of failing or dropping out of a course), and thirdly adapting or choosing course material according to the students' needs [85, 181]. Yet, there is no consensus on whether such applications have a positive effect on learning in general. Ferguson et al. conclude (in 2016) that there is "no overwhelming evidence that learning analytics have fostered more effective and efficient learning processes and organizations."[85, p. 9], while admitting that this might be due to the youth of the field which originated between 2010 and 2011.

Many reports in the Learning Analytics literature suggest statistical methods for behaviour prediction without putting these to a pedagogic use [92, 86]. Commonly predicted learning behaviours are academic performance and dropout (from a course or course of study). Section 5.1 and Section 5.2 define and evaluate predictors of these behaviours, and review the literature concerning these types of predictions in greater detail.

Learning Analytics are often presented within Learning Management Systems in so-called "analytics dashboards". Sometimes, only the teachers of a course are provided with such a dashboard to allow progress monitoring [248]. Yet, students have been shown to benefit from such presentations, for instance with improved self-assessments [131] and increased motivation [49]. Informing students about their participation behaviour in online discussions improved their mode of participation [261], and informing students about the prior knowledge of their peers improved collaboration effectiveness [219]. Park et al. found that an introduced analytics dashboard, while positively received by the students, who reported an improved learning behaviour, did not improve examination outcomes significantly [185].

Apart from providing analytics dashboards, identifying "at-risk" students in order to implement interventions is a common use of learning analytics [141, 144]. Here, "at-risk"

usually refers to a predicted dropout or failing of a course. Interventions often encompass proactively warning the student, and initializing personal communication with the teacher. Cambruzzi et al. let teachers contact students who were identified to match a "drop-out profile" by a learning analytics system, which resulted in higher retention [37]. Similarly, Arnold et al. increased retention and examination success by sending personalized emails containing performance predictions which were encoded as a traffic light signal. Often, students perceived these emails as a personal communication from the teacher even though they were computer-generated. Choi et al. identified at-risk students by analyzing data obtained in an audience response system and used a combination of automatically sent emails and personal communication (including phone calls) which decreased dropout [43]. It is noteworthy that teacher support *alone* can increase retention: Onah et al. report that students taught in a MOOC with increased teacher contact exhibited lower dropout rates [176], and a similar finding was published by Holmgren and Johannson [113].

3.2 Pervasive and Collaborative Learning Technologies

In this section, literature regarding the effects of educational software on Self-Regulated Learning, collaboration, and the learning of STEM's formal languages is reviewed. Research in these fields is often conducted in laboratory studies and is often rooted in specific learning theories or models.

Self-Regulated Learning. Self-regulated learning theory describes how students "personally activate, alter, and sustain their learning practices in specific contexts"[267, p. 307], and self-regulation is often stressed to be necessary for students both in tertiary education [163], and online or blended learning [129].

As most of tertiary education today is a form of blended or online learning [56], it seems appropriate to examine how software can foster self-regulation. Dabbagh and Kitsantas suggest a pedagogical framework on how *existing* social web applications (such as a blog software) can be used to sustain self-regulation, by analysing different uses of such software: For instance, an instructor can encourage students to use a blog as a private learning journal (the use being personal information management), or to visit and comment in blogs of their peers (the use being collaborative learning), or to use tools such as RSS feeds to aggregate blog posts (the use being information aggregation) [55]. This illustrates that students have access to a large variety of software applications which are *potentially* suited to sustain self-regulation, while initially not being conceived as educational software. Arguably, such uses require technical knowledge, and (to some extent) creativity from the learners.

Other research focusses on examining software explicitly designed to sustain self-regulated learning. A challenge mentioned in the literature is the measurement of self-regulated behaviours, which is often assessed using questionnaires or structured interviews [246]. An approach to *automatic* assessment of self-regulated learning behaviour is suggested by Hadwin et al. who used extensive analyses of trace data recorded in a controlled experiment in which students used a note-taking and annotation tool [99]. Yet, no correlation of *reported* and *traced* self-regulated learning behaviour could be found, which arguably illustrates that such an automatic assessment is difficult to realize.

Stage models of self-regulated learning structure the learning process in a repeating sequence of stages the learner enacts while learning.

One commonly used model is Winne's four-stage model [260], with the four stages being:

1. "Task definition", in which the learner conceptualizes the task to be performed.
2. "Goal-setting or planning", in which the learner chooses strategies they want to carry out.
3. "Enactment", in which the strategies from phase two are used.
4. "Adaption", in which the learner reflects on their learning.

One approach to foster self-regulated learning with educational software is to address specific phases (such as supporting the planning phase by providing a schedule application) [59].

Van Lear and Elen evaluated 95 reports on self-regulation in blended learning and found seven attributes common for software applications which fostered self-regulation [246]:

1. Authenticity: The application provides "real-world" and relevant practice or shows connections to "real-life" situations.
2. Personalization: The application adapts to the need of the learner (ranging from name-recognition to adapting to the performance level).
3. Learner-control: The application allows the learner to make relevant decisions (such as controlling pacing, choosing content or content sequencing, etc.).
4. Scaffolding: The application supports learners in performing tasks that would have been out of their reach otherwise, with the support being reduced (faded) over time.
5. Interaction: The application provides interactions with the teacher or peers or provides interactive learning material.
6. Reflection: The application prompts the learner to reflect on their learning.
7. Calibration: The application allows the learner to compare actual to perceived achievement (for instance by providing assessments).

Van Lear and Elen argue that these attributes can be related to different phases of the four-stage model, for instance, authenticity and personalization features help to conceptualize (phase one) and so on.

Another branch of self-regulated learning research, which has only been briefly touched on by Van Lear and Stijn (by examining the interactivity between peers) is co-regulation, that is regulation occurring *between* learners. Kaplan makes suggestions about how to foster co-regulation in an LMS [127]; For instance by providing co-assessment alongside self-assessment functionalities. Crough and Christopher increased self-regulated learning and engagement by integrating co-regulative processes (collaboratively choosing material and devising suggestions for examination questions) into a second-year biochemistry course [54].

Collaborative learning. Computer-Supported Collaborative Learning (CSCL) theory describes how software can sustain collaborative learning. Stahl [232] builds this theory around four concepts which can each be supported by software: *Collaborative knowledge building* through interactions and discourse between learners (supported for instance by discussion forums), *individual and group perspectives* which result from these interactions, *mediating artefacts* which are represented by software (for instance a physics simulation), and *interaction analysis* which refers to the (possibly software-driven) evaluation of interactions to draw conclusions about the learning progress.

An example of software-driven interaction analysis is given by Sundararajan, who evaluated online discussions in LMSs using social network analysis (analysing graph representations

of interactions between learners) and found that students whose knowledge increased also became more prominent (more central) in the social network of the course. Indeed, these students functioned as "knowledge mediators" [236].

While the focus in the last decades lay mainly on software to sustain "collaborative knowledge construction" (for instance in online forums or chats), more recent approaches focus on regulation in CSCL [120]. In general, collaborative learning demands both self-regulation and co-regulation from the students [165, 120]. An example for software applications which foster self- and co-regulation are group awareness tools, which provide its users with information about the other users' knowledge or ability or the group progress as a whole [165]. Sangin et al. prompted students with assessments of their peers' prior knowledge in an online learning environment and found an improved collaboration effectiveness and improved learning outcomes [219].

A rather strict form of regulation is provided by computer-supported collaboration scripts, which guide interactions between learners. Usually, the role of the computer within such scripts is to prompt students with specific tasks or to keep track of the overall progress of a class. In the MURDER script, pairs of learners review text passages where each learner assumes either the role of the summarizer (who summarizes the passage) or a listener (who detects errors in the other's summary)[137]. In the Concept Grid script [66], each member of a group gives a short definition of a term they are tasked to research. Afterwards, the definitions are placed on a map, and descriptions of the relationships between neighbouring concepts are elaborated collaboratively.

It is worth noting that both the literature on "group awareness tools" and especially on collaboration scripts evaluates learning in small groups (usually ranging from 2 to 5 students) within relatively small classes [50]. In tertiary mass education, with class sizes of a few hundred students, collaboration (if not further scripted) might more resemble the "mass collaboration" of platforms like Wikipedia, where relatively few users perform the majority of actions [51].

Formal languages and Microworlds. The discourse in STEM fields is replete with the use of formal languages, and software seems to be suited to ease the learning and use of these languages. Consequently, there is a plethora of learning tools aimed specifically at conveying formal languages, which are often provided in a stand-alone web application, or are integrated into online homework systems (see [197] and [128] for examples).

These tools are often referred to as "editors", "verifiers" or "checkers", or "development environments". Three major uses of such tools can be identified:

- **Program evaluation**: The software evaluates source code written in a programming language and prompts the learner with the results. Languages from various programming paradigms have been supported with such systems: Ranging from functional programming [5] and logic programming [5] to imperative object-oriented programming [128]. Mini-languages, also referred to as pedagogic languages, which usually only comprise a small set of syntactic elements and are conceived for programming beginners, also fall under this category [32]. Examples of mini-languages are robot Karel 3D [244], turtle graphics (implemented in the programming language Logo) [230], and scratch [158].
- **Artefact Verification**: The software verifies the *semantic correctness* of an artefact written in a formal language. This is especially interesting in mathematics and logic education. Examples of such "verified artefacts" are proofs in formal logic [112, 235] or term manipulations in algebra [250].

- **Simulation**: The software simulates a virtual machine given an operation code and displays the operation of the machine. Examples are assembly code computers [251, 213], or various automata used in theoretical computer science (finite automata, pushdown automata and Turing machines in various forms) [212].

Note that this categorization is not strict: Mini languages like Karel 3D [244] are often used to simulate agents (robots, turtles, etc.), and could, therefore, be categorized as "simulations". Software which allows writing and running assembly code could be categorized under "program evaluation". There are applications which combine aspects of several categories such as LLparse and LRparse [212], which are used to teach formal grammars (the software *verifies* aspects of the grammar, and *simulates* a parsing machine afterwards).

Often, blocks are used to represent formal languages, where the elements of a language (such as keywords in programming languages) are represented by user interface elements and can be manipulated by the user [207, 28, 84]. Block languages are often used for programming beginners as it has been argued that simultaneously mastering syntax and semantics of a programming language is a strong hurdle for beginners [96, 122]. Yet, block representations can also be found in mathematical applications, where parts of a formula or parts of a proof are represented as blocks [112, 235].

All of these applications can be regarded as "microworlds", defined as "open-ended exploratory computer environments" [76, p.1]. Microworlds usually consist of objects which can be manipulated according to given rules, and often encompass multiple representations of these objects [76]. Such applications have been successfully used in teaching various STEM fields [121], and have been praised, among others, because they enable playful exploration of a domain [209].

In the examples discussed above, the domains to be explored by the learners are formal languages (operation codes, programming languages, formal proofs, etc.) with the syntax and semantics of these languages defining the rules within the domain. From a Computer-Supported Collaborative Learning perspective as proposed by Stahl [232], these tools are media by which collaboration can take place (for instance when peers discuss why a program works, or does not work, as intended).

3.3 Learning Theories in Technology-Enhanced Learning

Design decisions made for the implementation of Backstage 2 / Projects were (often) informed by learning theories, and the following section discusses briefly the main influences for this work, namely the conceptual change model and a group of hierarchical learning models (where learning is grouped in stages which have to be consecutively mastered to advance).

The over-arching learning theory behind these models is Constructivism, a learning theory which emerged in the 20th century and is very prevalent today's discourse [126]. Constructivism models learning as the active construction of knowledge by the learner through interaction with other learners or teachers (referred to as *social* Constructivism, pioneered by Vygotsky) or through active experience (referred to as *cognitive* Constructivism, pioneered by Piaget) [126].

The field of Technology-Enhanced Learning, while not following a coherent learning theory, has produced certain learning theories in their own right, which have been largely influenced by the constructivist mindset: Computer-Supported Collaborative Learning theory (which was briefly discussed in the last section), models learning as a constructive process between learners which is mediated by a software artefact (such as a physics simulation students can

discuss and explore) [232]. Online Collaborative Learning theory, introduced by Harasism, focusses on learning in online environments through group discussion, collaboration, and conceptual change [100], concepts which prominent in constructivism. Arguably, the connectivist learning theory introduced by Siemens, which models learning as an exchange of information within a community [94], was also influenced by social constructivism.

The cognitivist learning theory, which views learning as processing and organization of information, has influenced many *concrete* learning software projects such as the first intelligent tutor systems, or knowledge organization tools (such as mind mapping applications) [152]. Educational software is praised, among others, for its capacity to reduce the learners' cognitive load [71], and Cognitive Load Theory, which describes how different mental efforts influence learning, originated in cognitivist ideas [152].

It has to be noted that there is a difference between a learning theory and a pedagogy *informed* by a learning theory. In the constructivist model, for instance, all knowledge has to be *constructed* by the learner, but this does not mean that the teacher has to enforce constant collaboration or exploration on the learners' side. Indeed, actively listening to a lecture is a form of constructivist learning (while lecturing is not necessarily a form of constructivist pedagogy).

Conceptual Change and Systematic Errors The constructivist learning theory describes the state of "disequilibrium", as a state a learner can be in when encountering new information which conflicts with their perceptions [126]. By resolving this conflict, (by "equilibrating") a learner integrates the new information into their perceptions which might require them to rebuild ("accommodate") parts of their knowledge structures. This, arguably theoretical, view on learning has been supported by numerous observations in science education. Here, students often have misconceptions (also referred to as naïve concepts or preconcepts) of a scientific phenomenon. Learning in these cases requires students to undergo "conceptual change": Abolishing the misconception and accommodating the "correct" concept. Student misconceptions and conceptual change are typically researched in secondary science education [45, 192].

Systematic errors, which can be defined as "erroneous, non-random applications of beliefs" [45, p. 33], are typically discussed with a cognitivist mindset (information being erroneously interpreted, or internal organizations failing to produce the correct output) and are usually discussed in mathematics and computer science education literature [45]. Several attempts have been made to classify systematic errors in student responses in general categories (see for instance [200] and [168]), resulting in error categories like "errors due to deficient mastery of prerequisite skills"[200]. More recently, Rakes and Ronau classified erroneous student responses in mathematics and evaluated the data using structural equation modelling, discovering strong interrelationships between errors in different content domains (such as geometry and algebra) [201].

While misconceptions are usually related to *concrete* phenomena and systematic errors usually refer to *abstractions* (like algorithms), both are very similar concepts, and recently, authors have started to discuss learning problems caused by misconceptions in the computer science and mathematics literature [198, 201].

Regardless of the terminology used, or the learning theoretical background of the teacher, systematic errors and misconceptions are known to be particularly robust against "traditional" instruction [45]. Posner et al. suggested that students abolish old (even if erroneous) concepts only if they are perceived as inadequate while new concepts are perceived as intelligible, plausible and fruitful [192]. Hence, teaching methods fostering conceptual change usually involve face-to-face instruction where the teacher induces an internal conflict (the afore-mentioned disequilibrium) of the students with their conceptions [224]. An arguably more scalable approach are refutation texts which usually contain an argument against a common misconception [240].

Hierarchical Learning Models Several models propose hierarchies of skills of learner proficiency. In these models, "higher-order skills" can only be obtained when the student masters "lower-order skills". This is relevant for learning software, as such models can allow modelling behaviour which is natural for "human" tutors: Some exercises can only be tasked if lower-order skills are known to be sufficiently mastered.

A prominent example of such a learning model is Fisher's Skill Theory, according to which learners navigate a hierarchical framework of skills where high-level skills depend on lower-level skills [236].

In the literature on problem-solving, hierarchical models are more concrete: Newman's Hierarchical Error Model describes the process of problem-solving in a hierarchy of steps [170], similar to the Hierarchical Model of Programming Skill Acquisition proposed by Lopez et al. [150]. Arguably, programming is related to problem-solving. In both models, the first steps are concerned with "understanding the problem" (encompassing capabilities in a natural or a programming language).

Other learning models *imply* a hierarchy of skills: The Conscious-Competence Model proposed by Burch, organizes learning in four phases, the first of which consists in recognizing what the learner is yet incapable of doing [35]. The Kruger Dunning effect, famously summarized by the phrase "we argue that the skills that engender competence in a particular domain are often the very same skills necessary to evaluate competence in that domain"[140, p.1] implies a similar hierarchy.

CHAPTER 4

Technology-Enhanced Learning and Teaching Formats

This chapter describes the design principles and components of the educational software prototype Backstage 2 / Projects, the functionalities it provides to learners and teachers, the learning and teaching formats it enables, and the evaluation of these formats. For brevity, "learning and teaching formats" are referred to simply as "teaching formats" in the following.

Teaching formats. There is no commonly accepted definition of the term "teaching format", which is frequently used to describe the general organization of a course: Several authors report on the integration of different teaching methods, such as group work or experimental tasks, in a course format which would traditionally only encompass a didactic lecture [36, 3, 89]. According to this practice, we loosely define a teaching and learning format as the "long term" or "general" organization of teaching methods within a course. Note that *teaching methods*, defined as "a set of principles, procedures or strategies to be implemented by teachers to achieve desired learning in students" [256, p. v], usually describe the interactions within a single lesson, while *teaching formats* span over several lessons. In that sense, learning and teaching formats are similar to Dillenbourg's definition of "macro scripts", while "teaching methods" are similar to Dillenbourg's "micro scripts" [68].

Combining teaching methods to realize teaching formats is a common practice: A computer science course typically encompasses lectures (introducing new concepts) and lab sessions or homework (for putting the concepts introduced in the lectures to use), a seminar may encompass an oral presentation of a topic as well as a written essay.

Enhanced formats and enhanced methods. Software is often used to improve teaching formats by improving the teaching format's *teaching methods*, for example by making these teaching methods suitable for large classes: Digital backchannels allow discussions within large lectures that would be impossible otherwise, audience response systems allow conducting quizzes and get assessments in large lectures, and virtual laboratories can give large numbers of students access to experiments which are too costly to be provided in a traditional, non-mediated form.

This approach of solving problems of large-class teaching by improving teaching methods with technology seems common: Bower et al. found that most technology-enhanced learning design frameworks (i.e. sets of general guidelines to help teachers create technology-

supported learning scenarios) gave indications how to design learning tasks, while few were concerned with course formats [29].

A teaching format may be considered as "technology-enhanced" if it encompasses at least one "technology-enhanced" teaching method, yet software can improve a format as a whole by connecting its components: Information gathered while teaching with one method can be used to improve teaching with other methods. For instance, information on learning problems or common errors detected in homework submissions can be prompted to the lecturer who can then address these problems upcoming lectures. The formats described in the following subsections use this approach of transferring information between teaching methods to some extend, yet its potential goes beyond these few examples. A perspective on teaching formats relying on information transfer between teaching methods, which could be realized relatively easily with the present software, can be found in Section 7.2.

Homework and feedback. The formats discussed in the following focus on asynchronous learning, in which homework assignments and feedback (provided automatically or by teachers) play an important role. Findings on the effectiveness of homework are mixed. In a popular meta-study, Cooper found that there were generally positive effects of homework on achievement while unveiling an influence effect of student age on the effectiveness of homework: In primary education positive effects were near zero, yet the positive effects grew according to student age with the largest effects being measured for high-school students of grades 10 to 12 [46]. This observation has been reproduced in more recent meta-studies [47] and [101, Chap. 10]. Although these findings stem from secondary education research, they are considered of importance here, as effects holding for older students in secondary education can be expected to hold for university students in their first terms.

Published evaluations of homework effectiveness in tertiary STEM education are less common than reports on primary and secondary education, yet assigning homework exercises is a common practice in these fields. Here, homework typically requires students to make use of freshly acquired techniques. Often, online homework systems are used to manage assignments and to provide automated feedback (see [180] for examples in statistics, [30] for mathematics, [254] for physics, and [60] for computer science). The formats introduced in the following rely on similar functionalities.

Several authors criticize research methods of studies on homework efficacy [46, 101, 242], often pointing out the that large variety of actual practices of *implementing homework* can hardly be generalized, and because the actual effectiveness may be caused by secondary effects and not by the task of doing homework itself. Indeed, Zimmerman and Kitsantas found that doing homework had an impact on the students' self-efficacy and perceived responsibility, which in turn influenced academic achievement [268], and Cooper already noted in the first meta-analysis that it presumably makes a large difference for learning whether the teacher just collects homework, grades it, or gives detailed feedback [46].

Feedback, for that matter, is known to have one of the largest impacts on learning (established for instance by Hattie in his "visible learning" study [101], or in [102]). Yet, in tertiary education, giving elaborated and directive feedback (as Hattie suggests), is difficult, and large classes and limited teacher time are often cited as causes of imprecise or shallow feedback [171, 38]. An interesting branch of research in higher education focusses on the inability of certain students to put the received feedback (even if of high quality) to use, a phenomenon referred to as the "feedback gap" [80]. Different causes for this problem are discussed in the literature, among others difficulties in the subject matter [80], attitudinal and motivational factors [264], and pedagogical literacy [195]. To "bridge that gap", feedback dialogues between teacher and student (in contrast to one-way feedback provided by the teacher) are often suggested [171, 195, 38].

The formats presented in this section each try to improve certain aspects of homework and feedback: Reducing workload when revising homework (Section 4.2), nudging students to do homework (Section 4.3), evaluating peer feedback on progamming homework (Section 4.4), and facilitating the use of formal languages with automated feedback (Section 4.5).

Teaching formats built from components. Often, the same educational software can be used for different (educational) purposes. Distributing learning material can be used to support a flipped classroom, as well as distributing additional sources. Online-quizzes can be published in the weeks before the examination for recapitulation, or to spark a discussion in class. It is then the teacher's task to devise a format that suits the intended learning outcomes, which can encompass choosing the appropriate software.

There seems to be an established set of "common" functionalities which are found in most available Learning Management Systems (LMSs). In the scope for this research, a set of 86 currently available LMSs were categorized by the functionalities they provided [110]. We found that all examined LMSs provided functionalities for document management (which is not surprising), and most of them allowed workflow management (such as assigning exercises) and provided some kind of interactivity (such as conducting quizzes). Interestingly, functionalities for student collaboration and communication were less common. Also, all functionalities provided by the examined LMSs could be classified under five categories (namely: Document Management, Workflow Management, Content Enrichment, Input Interactions, and Learning Analytics), with no functionalities being unclassifiable under these categories [110].

Backstage 2 / Projects is in that sense a Learning Management System (providing many of the "common" functionalities) specialized on conducting asynchronous learning formats. The following section describes the functionalities of the software in general and the subsequent sections describe how these functionalities were used to define and conduct teaching formats.

Results on the effectiveness of these formats in improving learning outcomes or behaviours are discussed in Chapter 6. Evaluation results regarding the students' attitude towards these formats, as well as general observations, are reported in this Chapter. Some formats rely on predictive learning analytics which are computed while a course is carried out. In these cases, the predictors are briefly described and a reference to the sections discussing the implementation and quality of these predictors are given.

4.1 Software Components and Functionalities

The basic building block for learning formats with Backstage 2 / Projects are projects. Every user of the platform can start a project and invite other users to join. A project is simply a collection of at least one participant, a set of documents, a set of assignments, a set of subprojects, and a schedule. Note that the term "project" is used in a broad sense: A project is a collaboration between participants which is supported by the software with functionalities to organize and author documents, assign tasks, and schedule events.

For the sake of completeness it has to be noted, that Backstage's projects can be used to enact Project-Based Learning (PBL): A teaching format in which students collaborate over a longer period and "apply what they know to solve authentic problems and produce results that matter" [159, p. 2], with the exact results not being specified as the project starts. The software was used to organize PBL in several instances, yet this use did not address concrete problems of tertiary mass education as the other formats introduced in the following sections. Therefore, reports on using the software for PBL is not included in this report.

https://chat.pms.ifi.lmu.de

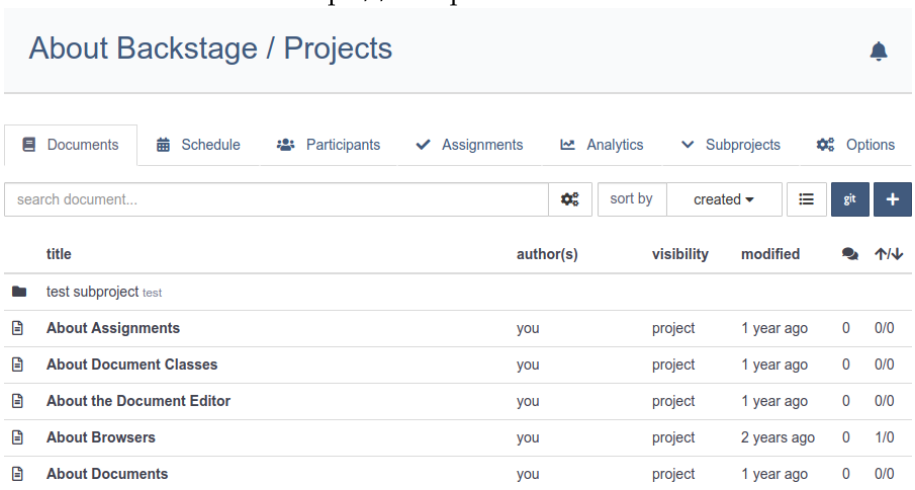


Figure 4.1: Project navigation and document view of the project "About Backstage Projects".

Figure 4.1 shows the main view and the documents of a project titled "About Backstage / Projects", which was (and is at the time of writing this report) used to document the software. The navigation bar (below the project title) allows navigating the different views of the project via tabs. Often, projects do not make use of all the provided functionalities (the documentation project, for instance, has no scheduled dates, and no assignments), in which case the regarding tabs are hidden from the navigation view.[1]

Several project views encompass lists of objects (lists of documents, assignments, participants and subprojects). In these cases the regarding view is always organized similar to the document view seen in Figure 4.1: A list of the objects in question is displayed below by a search bar and an add-button (to invite a participant, to add a document, etc.).

Documents. Documents can be added by all participants of a project. They play a central role for all teaching formats presented in the following, as they can be used for a multitude of purposes: Documents can contain homework submissions, teaching material and supplementary material, student questions, etc.

There are three different types of documents: Code documents (containing source code of programming languages or other formal languages), text documents containing text following the markdown syntax[2] which are automatically rendered, and PDF documents. The platform allows users to run code documents provided by any user directly in the browser.

Documents can have different visibilities: Private (only visible to the authors of a document), teacher (only visible for teachers and the author), project (only visible to the members of the project), and public (visible for everyone, including all members of the parent project). Documents can be commented on, and every participant can comment on any document he or she can see. A document with an attached comment can be seen in Figure 4.2. Comments can also be commented on, following the same rules as commenting on a document.

An important functionality for documents is "labelling". With labelling, a label-document can be attached to labelled document, which is then displayed below it. Generally, labelling can be used if *one document* is relevant *for several* other documents. In this way, labelling is

[1] Figure 4.1 shows all possible tabs for the sake of completeness.
[2] `https://daringfireball.net/projects/markdown/`

Figure 4.2: A code document titled "Hello ICL" containing Haskell code (which was executed), and a comment.

used to attach descriptions of systematic errors to homework submissions which contain that error (described in Section 4.2), and to assess the quality of homework submissions (described in Section 4.3). Generally, document labelling is similar to commenting, as it attaches further information to a document. Yet, different from comments, labels can be attached to any number of documents, and changes to a label are reflected in all labelled documents. Consider for instance a label which contains the assessment that a subject was not well understood and needs to be studied more thoroughly. Several students might find this label attached to their homework. Students might then want to have additional (or alternative) material on the subject, and references to this material can then be added by *all members of a project* (for instance through comments on the label) and little by little as needed. Currently, labelling is only available to teachers.

Schedule. A project's schedule is simply a set of dates encompassing description texts, which are displayed in a calendar view. Currently, only teachers can add dates to a schedule.

Participants. Each participant of a project has a role, either teacher, learner or administrator. Teachers and administrators may change project settings and pose assignments. For technical reasons, administrators are the only participants who are allowed to remove members or documents from a project. The system guarantees that there is always at least one administrator in each project.

Assignments. Assignments encapsulate actions to be performed by a project participant of any role. They consist of a *context* (telling the member what to do and providing the necessary information), an *artefact* (which is the outcome of the performed action), and an optional due date. This system allows to guide a large variety of actions performed on the platform: If, for example, a student is tasked to provide a solution to a homework exercise, the exercise text (a document containing the problem statement) is part of the assignment's *context*, and their submission (a document containing the solution) is the *artefact*. If a teacher is tasked to provide feedback on that student's submission in the following assignment, the student's submission is that assignment's context, and the teacher's review (typically in form of a comment) is the artefact.

This structure is reflected in the assignment view: The left-hand side of this view always displays the context, while the right-hand side always displays the artefact of an assignment. An example assignment can be seen in Figure 4.3: The right-hand side of the interface

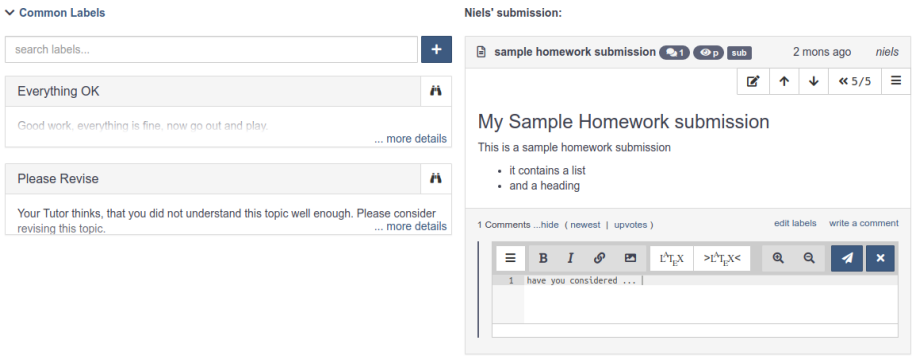


Figure 4.3: An assignment review a sample homework submission by the user Niels. The left-hand side displays "labels" which can be used to annotate the submission.

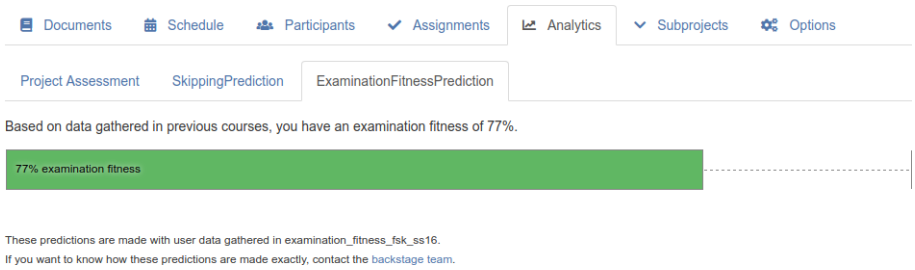


Figure 4.4: Examination fitness prediction for a student.

shows the assignment's artefact (the comment given by the reviewer below the reviewed document) and the left-hand side shows the context (here a list of common labels which can be attached to the reviewed document). The context of a review assignment can also contain other additional material such as exemplary solutions.

Learning Analytics. The platform allows teachers and administrators to add a set of predictors to a project. A predictor consists of a set of training datasets (which are typically gathered in previous projects), for each participant a set of observations (that were gathered while the course is carried out), and for each participant a prediction. For example, the examination fitness predictor holds for each learner in a project an estimation of their examination outcome in the form of a value between 0% and 100%. This prediction is based on the numbers of submitted homework assignments and submission qualities.

Figure 4.4 shows the student view of their current examination fitness prediction. Currently, teachers and administrators of a project can see all predictions for all students and can specify the training datasets the predictions are based upon, while learners only see their own predictions. Section 5 describes the predictors, their definitions and qualities in greater detail.

Subprojects. A project can encompass several subprojects, which may serve different purposes: For instance, they can organize homework assignments by topic, or organize student teams for group work. The depth of project nesting is not limited by the software. Data relevant for behaviour prediction that is gathered within a subproject is stored in the subproject if it has a fitting predictor, in the deepest parent project holding a fitting predictor if the subproject has no such predictor, or not at all if no such parent project exists.

Options. Users can configure certain aspects of the projects they are in, such as whether, and when, they want to receive news emails on the latest activity in the project, or whether they want to be reminded of assignment deadlines via email.

Newsfeed. Actions performed within a project (such as adding or commenting on a document) are logged by the system, and displayed in the project newsfeed. News can be searched, and by clicking on an item in the newsfeed the platform navigates to the regarding item (for instance, clicking on the message that a user has commented on a document navigates to that document).

The orchestration mechanism. Orchestration is referred to as "the design and real-time management of multiple classroom activities, various learning processes and numerous teaching actions" [67, p. 3]. While in this definition technology is not mentioned, it plays a large role in many "orchestrated designs" found in the literature. In these designs, the emphasis lies on lesson structuring and time-management within lessons with the help of technology. A formalized approach to classroom orchestration are Computer-Supported Collaboration Scripts (CSCSs). CSCSs enable collaborative learning by organizing learning activities, and the roles learners take within these activities, in sequences of phases [65, 137], while focussing on group formation and organization. Note that the term "orchestration" implies a certain teacher-centeredness, as it implies that a centralized conductor (arguably the teacher) organizes the learning [67]. Yet, teacher orchestration and (student-centred) collaborative learning scripts are usually carried out together, with the students carrying out different learning activities in different phases, and the teacher overseeing the learning, and starting and ending the phases.

While the literature on orchestration technologies and CSCSs focusses on *synchronous* learning, the orchestration mechanism of Backstage 2 Projects allows the organisation of *asynchronous* learning. In contrast to CSCSs, the mechanism introduced in the following focusses not on learning activities, but on the artefacts produced as *a result of* learning activities. When students (or teachers) are tasked to author a document, write a comment, or to start a project, the resulting artefacts are the document, the comment or the project. In this model, an assignment encapsulates the process of producing an artefact.

The orchestration mechanism allows the definition of orchestration scripts, which generate sequences of assignments, usually worked on by different actors, where the next assignment in a sequence depends on the artefact produced by the last assignment in the sequence. In the homework-review example described above, the orchestration mechanism would create an assignment for each homework exercise and each learner in a project, wait for their completion, generate a review assignment for each completed homework, and assign it to one of the project's teachers.

Arguably, this approach may appear quite technical on the cost of pedagogical reasoning: Approaches focussing on learning phases and activities seem to better reflect constructivist ideas, which are prevalent in today's pedagogical discourses. Yet, this approach holds several advantages for software design and for large class teaching formats:

- The mechanism realizes a **separation of concerns** as it only specifies task allocation (*what* is done by *whom*), and leaves task execution (*how* something is done) unspecified. This simplifies software design. Also, it takes into account that students and teachers have different preferred modes of working (especially if tasks are to performed asynchronously). In this way, the orchestration presented here reflects what Kobbe et al. refer to as "script mechanisms" as opposed to "script components" (which encompass for instance the learning activities) [137].
- The mechanism is **flexible with regard to exceptions** as scripted "decisions" are made for individual students. Consider for example a course in which a student needs

longer for a task than provided, and that the teacher grants a delay. In a phase-based approach, the "working phase" for that student would be extended, getting it of sync with the rest of the course. This might be undesirable because phases are arguably understood as guidances for complete courses, and it can be difficult to assess how future phases will be influenced by that decision. In an artefact-based approach, the realization of exceptions seems easier, as it requires simply changing one deadline. Also, it can be *inferred* directly from the script which depending assignments will be influenced by that decision (if the assignment allocations are not or not entirely based upon learning analytics). Another example of exceptions are failed assignments. Students might not meet deadlines (and a delay may be impossible to grant), leave the course, or fail for other reasons. The simple solution implemented here is that a failed assignment cannot lead to consecutive assignments (because no artefact was produced on which the next assignments would rely). While the unpredictability of classrooms is mentioned in the orchestration literature [215] and flexibility is praised in the CSCS literature [68], generic solutions to automatically react to such exceptions are (to the author's best knowledge) missing.

Note that scripts would need to halt over long periods of time while waiting for assignments to be completed. In order to prevent the software from blocking and to maintain code-readability, a mechanism similar to the "crash-and-rerun" model introduced by Little et al. in [149] was implemented. This model involves re-running a script several times until it completes, while relying on memoization: Tasks are only allocated if they have never been allocated before, with stored responses being used otherwise.

Scripts are executed in the following manner: The first assignment of a script (in the above example providing the solution for an exercise problem) is retrieved from the database. If the assignment failed or is not yet completed, nothing is done (the script evaluation crashes after step one). If the assignment as been completed (the student provided a solution), the next assignment in the script (the task to review the solution) is either retrieved from the database or assigned and stored if it did not yet exist. If it already existed and failed, or was not yet completed, nothing happens (the script crashes after step two). If it was completed, the next assignment is considered and so on. A script is then executed at regular intervals, either generating new assignments, crashing at the first failed or running assignment, or stopping at the script's last assignment.

This approach and the implemented API allow specifying orchestration scripts in a declarative manner. Listing 4.1 shows a simplified version of the homework-review orchestration script used in Sections 4.2 and 4.4. The orchestration component is implemented in the programming language Scala[3], and takes advantage of Scala's for-comprehensions[4] to run orchestration scripts. The current implementation also allows splitting execution chains, if more than one assignment should be assigned in parallel (for instance if more than one reviewer should review a solution as used in Section 4.4).

[3] `https://scala-lang.org`

[4] `https://docs.scala-lang.org/tour/for-comprehensions.html`

Listing 4.1: An orchestration script defining homework and review tasks. "solution" and "review" refer to the artefacts produced by the regarding tasks, "taskToReview" is a method generating a review task, and "informAboutReview" is a method sending a notifaction to the reviewed student.

```
//for all homework assignments a and a teacher t in the project
for {
        solution <- a
        review <- taskToReview(solution, t)
} yield {
        informAboutReview(...)
}
```

Note that, while modelling individual students as actors receiving assignments, this approach does explicitly include collaborative teaching formats: Review tasks can be assigned to peers instead of teachers (realizing peer review), and an assignment can encompass starting or joining a project with a number of peers (the artefact of this assignment being a project) which can be used to orchestrate group work.

As a last note on the orchestration component, consider the scalability of orchestration scripts defined in that manner. Many CSCSs in the literature include group formations with specific group sizes: Groups of 4 in the "Concept Grid script" [66], groups of 2 in the "MURDER script" [137], fixed group sizes determined by the course content in the "jigsaw script" [9]. The definitions of these scripts do not specify how groups should be formed if the number of students is not divisible by the specified group size, presumably because they are usually enacted in relatively small classes where workarounds can be found by the teacher "on the fly". For large classes of several hundred students, teachers could not possibly provide such solutions easily. When specifying an orchestration script as introduced above, the programmer would need to implement a certain flexibility in the group formation process (which can be realized with standard programming techniques), making it then scalable to arbitrary large classes. Dillenbourg identified a loss of flexibility as the main drawback of "computerized" scripts [66]. It might be added that flexibility has to be a property of a "computerized" script, the specification and implementation of which requires a suitable mechanism.

Eco-system and service architecture. Backstage 2 / Projects is a part of a larger eco-system and uses other software (both user interface components and web services) to provide its functionalities. Figure 4.5 illustrates the software components of the eco-system that are relevant for this report. All the displayed components, apart from the authentication service, were implemented either by computer science students in their bachelor or master theses, by Sebastian Mader who is currently working on the Backstage 2 research project, or by the author of this report.

There are three web-services on which the Backstage 2 / Projects relies: Firstly the service to create, read, update and delete (CRUD) documents of the platform, also providing document versioning which allows restoring previous versions of a document. Secondly, an instance of the open-source software keycloak[5] which manages authentication, login and logout as well as general user data. Thirdly, the web service "CoCoNut" (standing for concurrent code compile unit) is used to provide code compilation functionalities of code documents discussed above. The design of CoCoNut is introduced in the Elisabeth Lempa's bachelor thesis "Coconut 2: Concurrently Virtualising User Code Compilation" [146]. The teaching formats discussed in Section 4.4 and 4.5 largely rely on the use of this service.

[5] https://www.keycloak.org/

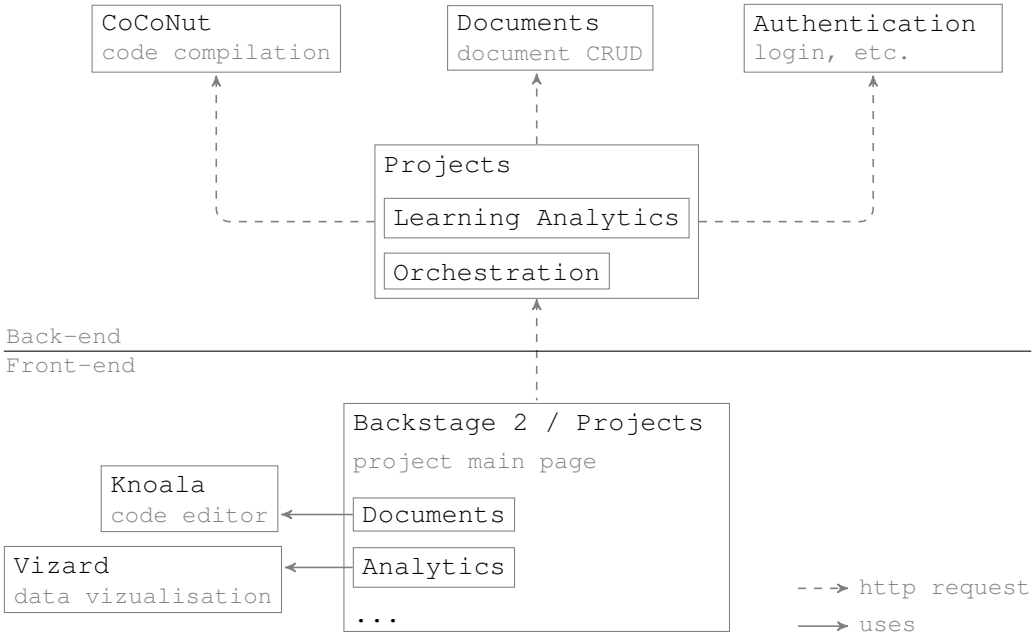


Figure 4.5: Service architecture of the Backstage 2. Front-end components (that can be displayed in a browser) are shown in the lower half, Back-end components (web-servers) are displayed in the upper half.

Two different front-end components are used throughout the application. Vizard is a software that is used for data visualization, for instance to display Learning Analytics, and Knoala (standing for Knowledgeable Other for Abstract Languages) is a text editor which is used to create and comment on documents on the platform, as well as providing the user interface for code compilation.

A big, yet unmentioned, component of the eco-system is called "Backstage 2 Courses": it is developed by Sebastian Mader [155] and is focussed mainly on synchronous teaching methods for large classes. While the scope of research of both systems is different, they share resources (such as the same webservices).

The following sections describe learning formats that can be built from the components.

4.2 Teacher Collaboration on Written Feedback

This learning and teaching format aims to improve teacher-written feedback while reducing the teachers' workload. The format was evaluated in a case study conducted in a course on theoretical computer science, the results of which can be found in [104]. This section summarizes the findings which regard student and teacher attitudes towards the format, while also discussing the pedagogical reasoning and perspectives for improvement.

Format definition. The teaching format uses the platform's assignments and orchestration mechanism to organize both homework delivered by students and homework correction performed by teachers. Assignments on different topics are grouped into subprojects which are used to schedule (typically weekly) homework assignments. The exercise texts, additional material, and the students' submissions are all represented as documents in the respective subprojects. Document comments can be used by students and teachers to discuss the teaching material, and by teachers to provide feedback on homework submissions.

Note that this layout allows to break the often criticized one-sided nature of feedback (see for instance [171]), as students can comment on the comments they received. An evaluation of this use can be found in Section 6.2, as it is not specific to this teaching format.

The format tries to reduce teachers' workload by allowing them to create notes about *prevalent errors and misconceptions* while revising homework, attach these notes to student submissions as feedback, reuse them for all submissions they revise, and (of course) share these notes with other teachers.

It is consistent with the platform design to store descriptions of misconceptions as documents: This allows their improvement over time, reusing them in future course venues, and allows students to publicly post questions about the presented error descriptions. As mentioned earlier, these documents are referred to as labels, and all labels of a project are part of a review assignment's context (depicted in Figure 4.3 in the previous section).

For the case study reported in the following, systematic errors which occurred in a previous course venue were known. This allowed providing the regarding label documents alongside the exercises as additional course material. This is referred to as *a priori* use as students could use descriptions of typical errors *before* submitting homework, while using labels to give feedback is referred to as *a posteriori* use. Students were more likely to avoid certain errors due to the *a priori* provision, an effect discussed in further detail in Section 6.1 and in [104]. Commonly, university courses are held at regular intervals (e.g. each year), encompassing similar exercises and materials. Therefore, providing students with common error descriptions from previous venues *a priori* is regarded as a part of this teaching format.

Note that the "intended" use of labels was to identify common errors and misconceptions, yet they are not restricted to that use. Indeed, they can be used for praise or for assessment (see Section 4.3), which motivates the generic term "label" used on the platform. In the format, teachers categorize student submissions while, at the same time, devising the categorization scheme (the most common errors). This collection of *error descriptions*, along with numbers of *error occurrences*, can provide valuable insights for teachers. It elicits which misconceptions remained after a lesson (or presumably were introduced by it), and may be used to test whether a new teaching method prevents misconceptions from occurring.

Preliminary results: Labelling validity. In a first evaluation, error occurrence rates were measured by examining homework submissions of a course on theoretical computer science. Figure 4.6 shows the results of evaluating 57 students' homework submitted in the second week of that course. Of 57 students, 46 made at least one error, with the most prevalent error occurring 27 times. The error distribution follows a power law: The four most prevalent errors constitute 47% of all errors found in the evaluated submissions, while about two-thirds of all errors occurred only once or twice.

This means that a software having stored descriptions of the four most common errors for these exercises could be useful in about half of all homework revisions.

A second validation was conducted to establish whether different teachers would be able to detect the same set of systematic errors in the same submissions when being provided with error descriptions (otherwise collaboration would be futile because different teachers would use error descriptions differently). Two measures were taken: Firstly, two teachers of the course were asked to record over four weeks the most common errors or misconceptions they found when revising exercises. Even though it was not specified how many misconceptions to record, both choose to report about 2 or 3 misconceptions per week. In two of the four weeks, the teachers agreed completely (set aside differences in phrasing), and in one week one teacher reported one misconception more than the other. Secondly, four other teachers were provided with descriptions of misconceptions previously gathered and asked to independently label a set of 20 homework submissions.

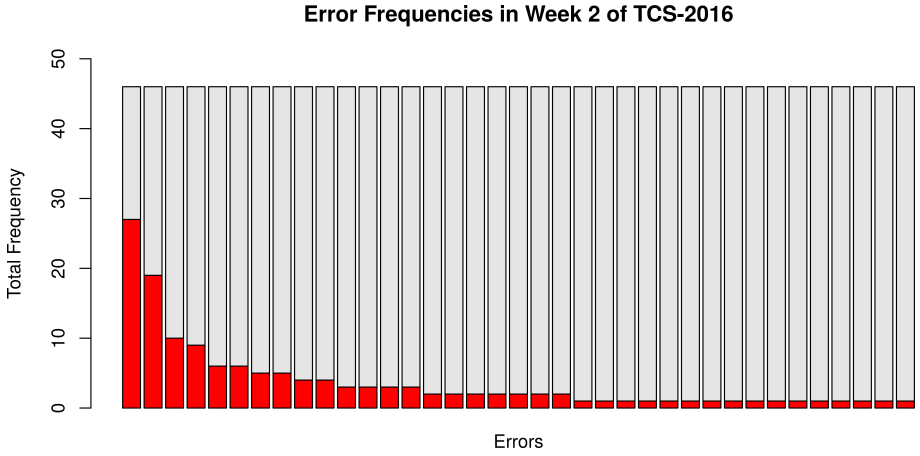


Figure 4.6: Distribution of systematic errors for the exercises of the second week of a course on theoretical computer science. One bar presents the occurrence frequency of one specific error. Figure adapted from [156]

To measure the inter-rater reliability the Fleiss-κ metric was used, which yielded an average value of 0.70 (which can regarded as a "substantial" agreement [143]) for all labelled misconceptions.

Case study results: Attitudes. The format was used in an introductory computer science course on the theory of computer science in the summer semester of 2018 at the author's university. The course lasted 14 weeks and encompassed 11 weekly homework assignments, each consisting of 3 to 4 exercises. The course's staff consisted of a professor, a teaching assistant, and 5 student tutors.

After the course had ended, students were asked to take an online survey on the course format in general, and the use of labels in particular. They were asked to indicate on a 6-point Likert scale ranging from "not at all" to "absolutely" whether they found the labels provided with the exercises helpful for their learning, whether they would be interested in knowing if peers made the same mistakes as they did, and if they found the labels helpful when received as feedback. The answers to these questions where somewhat ambiguous, as depicted in Figure 4.7 (left): Students found the *a priori* use more helpful than not, and the a posteriori use a bit less helpful. Answers regarding the interest of students in the errors made by their peers showed a large variance.

In an open item question for general remarks, some students praised the additional material provided by the labels.

In the questionnaire, students also indicated whether they used labels a priori while carrying out homework and whether they perceived personalized feedback as more or less helpful than feedback provided with labels. The results can be seen in Figure 4.7 (right): Most students who used labels *a priori* found both modes of feedback equally helpful, while most students who did not use them *a priori* perceived them as less helpful than personalized feedback. It has to be noted that the teachers in the course choose nearly always to provide an additional comment when attaching a label to a submission, often indicating that the error described in the label might be of concern for them.

Three of the four teachers of the course who had been using the labelling function for 11 weeks participated in a survey on their attitudes towards the system. They indicated that the functionalities reduced their workload and they found it helpful for revising homework. They also indicated that they would like to use such a system in the future and would share the systematic errors they found with other teachers.

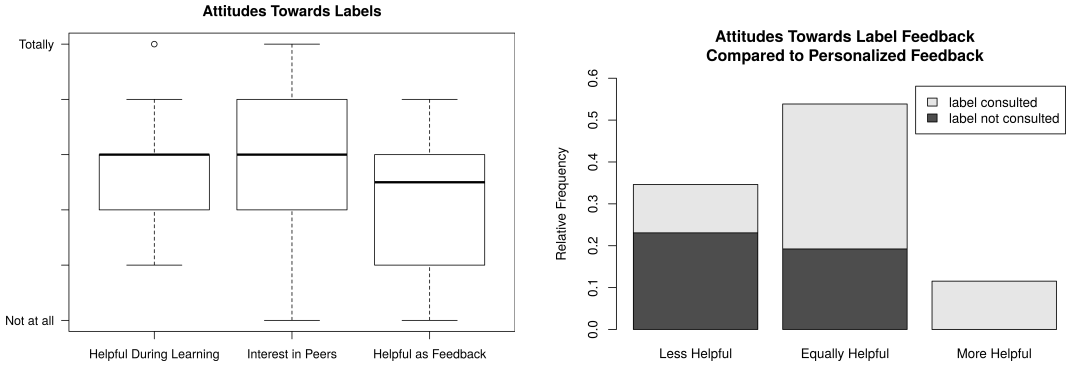


Figure 4.7: Left: Attitudes towards a priori and a posteriori use of labels. Right: Perceived helpfulness in comparison to personalized feedback and whether labels were used a priori. Figure adapted from [104].

Discussion and perspectives. It can be concluded that the format reaches its goals: Teachers reported the functionalities to be helpful, and the students were more or less ambivalent towards "labels as feedback" and a bit more favourable towards using labels as extra material while carrying out homework. Hence, the format introduces no negative aspects for the learners, while being positive for the teachers.

Though, it is unclear whether the format *improved* the feedback provided on the platform. Possibly, having the labels at hand encouraged the teachers to generally prefer labels over personalized comments which could have been more appropriate. On the other hand, compiling a set of common error descriptions and corrections in a text file, and pasting them as feedback when suitable, seems to be a common practice (at least at the author's faculty). With this regard, the reduced helpfulness of label-feedback perceived by certain students may be an illusion: What students perceive as "personalized" may just have been pasted from another text document.

It was found that students preferring personalized feedback over labels also did not use the labels *a priori* for their learning. Presumably, these students could not draw much help from the labels in either way, which seems related to the "feedback gap" discussed earlier. Note that this effect is believed to be (in part) caused by reduced academic literacy. This gives rise to an interesting software-based intervention: Software can detect whether a label document was examined by a student, or whether they deliberately avoid such documents. Additionally, students could be automatically asked if the content of such documents was comprehensible or useful for them. A teacher revising homework of a student who is known not to benefit from "label-feedback" could be advised to refrain from using labels for that student, to provide feedback in greater detail, or to seek direct contact with the student.

It has to be noted that this teaching format may not be applicable in all courses. Firstly, the subject matter might simply not cause systematic errors. Secondly, the occurrence distribution of errors is important for the format to work properly. Consider a previously unknown error which occurs in 15% of all submissions (which is among the highest rates of error occurrences found in the dataset) and that a teacher would need to encounter 3 instances of that error to classify it as systematic. Assuming a binary distribution of error occurrence (i.e. the submissions are not sorted by the errors they contain), it can be shown that at least 34 homework submissions have to be corrected by the teacher to be 90% sure to identify this error as systematic (respectively 52 submissions if the error occurs in 10% of all submissions). Hence this number of submissions has to be reached for the system to provide a benefit. While this may not seem like much, certain courses evaluated for this report did not meet this criterion (see the next section for an example).

An interesting perspective of this teaching format is its long-term use: In the evaluated case study, the label documents often contained mere error descriptions and explanations on how to correct these errors. In the future, further material could be added both by teachers and students: Counterexamples and worked examples of similar problems, links to additional material or exercises and so on. Also, such improvements could be guided to address material on the most prevalent errors first, as they will arguably have the greatest impact.

As a final remark on the perspectives of this teaching format, consider the following anecdote, which illustrates certain subtleties of teaching: In one venue (used to initially collect systematic errors and occurrence rates) a systematic error in the field of formal grammars was identified. Students were tasked to create a formal grammar describing a specified formal language. Many students submitted an unnecessarily complicated grammar using the Chomsky normal form[6]. This is not an error inherently, but nearly all students who tried to provide the grammar in normal form failed (presumably, because this form is not easy to master). In the subsequent venue (used to test whether the same systematic errors would reoccur), this error did not occur once. This was surprising, yet an explanation was easily found: In the first venue, the exercise introducing the normal form was provided *alongside* the exercise producing the error. In the second venue, the two exercises were given in subsequent weeks, due to organizational differences in the schedules. It is quite possible that many students were primed by the "normal form" exercise and used it unnecessarily in the other exercise. There are several effects known in the learning psychology literature which could explain this phenomenon: Namely priming [255] (exposition to a concept influences the responses towards a subsequently exposed concept) or retroactive inhibition [77, p.234 ff.] (learning of a new technique may inhibit the use of previously learned techniques). It was not the goal of this research to detect such effects, nor to reduce the occurrence of this particular error, yet its accidental observation is more than a curious side note: It exposes a direct effect of a teaching decision, and teaching formats which exploit teacher collaboration combined with statistical analysis might bring more such effects to light.

4.3 Analytics-based Nudging

Student passivity and class absenteeism are often mentioned in the literature on large class tertiary education [169, 253, 221, 214]. Related to these issues is insufficient homework completion, which can be regarded as a form of passivity. In evaluating homework completion rates at the author's faculty, it was found that students tend to *continue skipping* homework assignments after having skipped once (see Section 5.1 for figures on this observation). This relates passivity to course dropout: It is questionable whether a student neither attending a course's classes or other course-related venues nor submitting homework can be regarded as "taking the course". Indeed, in the literature on MOOCs (Massive Open Online Courses), where the problem of low course completion is often discussed [37, 263, 136, 132], dropout is commonly defined as "longer periods of inactivity" or as "discontinued participation".

The teaching format introduced in this section addresses the problem of low homework completion, which is less often discussed in the literature than passivity and class absenteeism. This under-representation might result from the difficulty to compare homework completion rates between courses, as they are influenced by course arrangements: If homework is compulsory or directly contributes to a course's final grade, completion rates can be expected to be higher than in a course with voluntary homework submission. By comparing different introductory computer science courses, Edgcomb et al. found homework completion rates

[6]The Wikipedia article defining the Chomsky normal form: `https://en.wikipedia.org/wiki/Chomsky_normal_form`, retrieved October 10th 2019

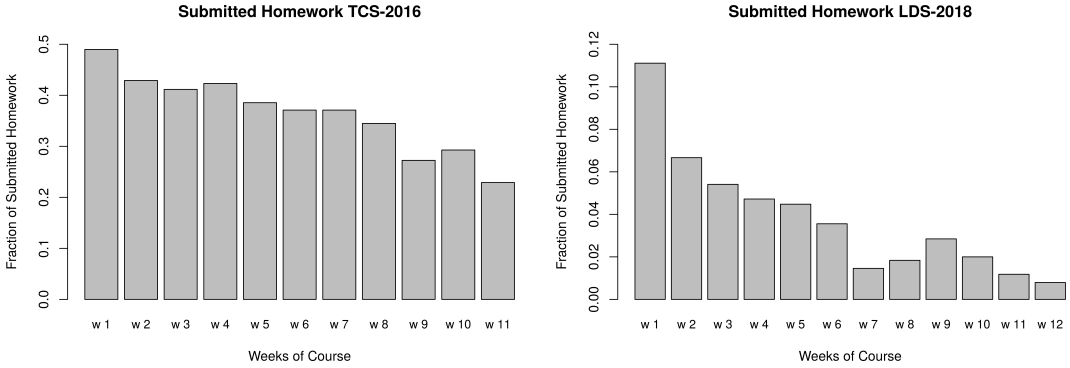


Figure 4.8: Development of homework completion rates for two computer science coures. Left: Theory of Computer Science in 2016 (TCS-2016) dropping from 49% to 21% completed homework assignments. Right: Logic and Discrete Structures in 2018 (LDM-2018) dropping from 11% to below 1% completed homework assignments. Note that in TCS-2016, homework completion was rewarded with up to 11% of the final grade, in LDM-2018 no rewards were given.

to vary as a function of grade points awarded for completing homework, with completion rates ranging from about 40% (in courses without compensation for homework) to near 100% (in a course awarding up to 30 of 100 available grade points) [75].

Figure 4.8 illustrates the development of homework completion rates throughout two computer science courses at the author's faculty: An introduction to theoretical computer science in 2016 (TCS) where each student's final grade was increased by 1% for each submitted homework, and a course on logic and discrete structures (LDM) in 2018, where no bonus was given. While completion rates in both courses differ largely in magnitude (TCS starting with 51% of homework being submitted in the first week, and LDM starting with 10%), a strong decrease in numbers of homework submissions can be easily identified throughout both courses.

The teaching format tries to increase homework completion rates by nudging the learners with individual predictions of their anticipated examination performances and risks of skipping (i.e., not handing in) their next assignments, to improve self-regulation and self-efficacy[7] which are known to positively influence homework completion [135, 19].

A "nudge" is defined by Thaler and Sunstein as "any aspect of the choice architecture that alters people's behaviour in a predictable way without forbidding any options" [239, p. 6], where "choice architecture" refers to the available options and their presentation to a choosing person. In education, nudges often take the form of deadlines, reminders, and "informational nudges" which aim to present important information in a salient way [57]. Interestingly, informational nudges, of which the nudging presented in the following is an example, are reported to have both positive and negative effects on task completion [57].

This teaching format encompasses sending automatically personalized emails to a course's students, a practice which has already been evaluated in the literature: Arnold et al. increased retention by informing students about their learning progress using a traffic light visualisation [8], and Lim et al. report an increased achievement and changes in learning patterns (related to Self-regulated Learning) for students who received personalized feedback emails from the OnTask[8] system [147].

[7] Self-efficacy refers to the personal judgement of "how well one can execute courses of action required to deal with prospective situations"[14].

[8] https://www.ontasklearning.org/

This section summarizes and extends the research presented in [106]. Results regarding changes of behaviour are discussed in greater detail in section 6.2, details on the implementation and the quality of the predictive Learning Analytics used in this course format are provided in Section 5.1 and Section 5.2.

Format definition. As in the format defined Section 4.2, assignments are used for weekly homework as well as to organize teacher reviews. Subprojects are used to group assignments, submissions and discussions on the several topics of a course.

Similar to the teaching format described in the last section, teachers use labels to categorize assignments. Yet, instead of using labels to organize and collaborate on descriptions of systematic errors, labels in this format are used to provide three assessment categories:

- **IK**, for insufficient knowledge, reflected by an incorrect use of symbols, statements like "I don't know how to solve this", or an answer not fitting the question, requiring the student to re-learn parts of the course.
- **OE**, for other errors, that is, errors not due to insufficient knowledge.
- **NE**, for no errors, otherwise.

A label document representing a category can contain additional information: For instance, an **IK**-label might list teaching material necessary for the completion of this topic's exercises, while an **NE**-label might provide material for further reading.

Sequences of categorized homework assignments are compiled for each of the course's students (encompassing an additional category **SKIP**, for skipped assignments). The software uses these sequences to compute predictions of examination results and risks of skipping the upcoming assignments, both represented by a number between zero and one. These predictions are then presented in two ways: Firstly, the predicted values are displayed in each students' analytics page (as seen in Figure 4.4) alongside changes in predictions since the most recent computation. Secondly, so-called "analytics reports" are sent regularly to all students of a course via e-mail. An example of such a report can be seen in Listing 4.2. It consists of three paragraphs which are adapted to the students' current predictions: The first paragraph reports the current examination fitness prediction and its change since the last computation. The second paragraph is only included if the student has, according to the analytics, a high risk of skipping and reports an estimated benefit of handing in the next assignment on the examination outcome. The third paragraph concludes the report with contact data of the course's teachers and instructions on how to unsubscribe analytics reports.

While the course is carried out, students provide (or choose not to provide) homework, which is then categorized by the teachers of the course. This categorization is similar to a grading (with three possible grades), which is a common task for teachers. Therefore, it can be argued that realizing this teaching format does not require much extra work form the course's teachers. The analytics used in this teaching format are based on the teachers' assessments – but these are not provided directly after the students handed in their assignments (teachers need time to make the assessments). This might seem counter-intuitive to the students, as personal predictions would change only when the teacher hands in their review. To make the system more responsive, unassessed submissions are counted as **OE** (other error), which is (in the data examined) by far the largest category of submissions. In this way, predictions change as students hand in homework submissions – yielding, for instance, higher examination fitness predictions.

Listing 4.2: Exemplary analytics report.

```
Dear <username>,

based on your previous assignments, we estimate your examination
fitness to be at 51%. This is 2% lower than our last estimation (
computed Tuesday the 11. of June). Please consider revising your
previous assignments.

You seem to have skipped one or several of your last assignments.
Consider that submitting your current assignments would increase
your estimated examination grade to 53%.

If you have problems with your assignments or further questions,
don't hesitate to contact one of your tutors:
...
If you don't want to receive any further analytics reports, you
can disable them in your personal options found at: <projecturl>/
options
Greetings and happy studying.
```

The predictors used in this teaching format depend on training data, which influence the predictor's properties. For instance, with the training data used for the experimental evaluation, skipping assignments and **IK**-assessments (insufficient knowledge) had negative effects on the examination fitness prediction (1.8% and 0.8 % respectively), while assessments of **OE** (other error) and **NE** (no error) had positive effects (of 0.7% and 0.3% respectively). Section 5.2 discusses these properties in greater detail. These values reflect the training data and seem reasonable: Skipping homework or making severe errors reduces the predicted examination outcome while making not-so-severe errors or no errors at all improves the prediction. Yet, with other courses, using different training data (possibly resulting in different predictions) might be more appropriate.

Note that using labels for quality assessment as presented above and for assessing systematic errors as described in the last section can be realized in the same course: In their current implementation, labels can represent a assessment category (for instance **IK**), and a systematic error (in this case caused by insufficient knowledge) at once. Teachers can even decide whether a systematic error belongs to a certain assessment category after the student submissions have been labelled, which would then change the currently computed predictions.

Results: Attitudes towards predictive analytics. The first evaluation of this course format was conducted in a bachelor degree computer science course on theoretical computer science in 2018.[9] At the time of that evaluation, the software did not provide analytics reports via e-mail, and the personal predictions visible to the students only included the current prediction, without indicating changes since the last estimation.

Nevertheless, students visited their personal analytics throughout the semester with notable peaks in visits at the beginning of the course and in the last weeks, which is illustrated in Figure 4.9. In average, students visited their personal analytics about once per week (median interval in days: 6.9, first quartile 2.7, third quartile 12.4), which was the interval with which predictions did change.

Student attitudes were measured in a survey at the end of the semester. Students indicated

[9] TCS-2018 see Apendix A.

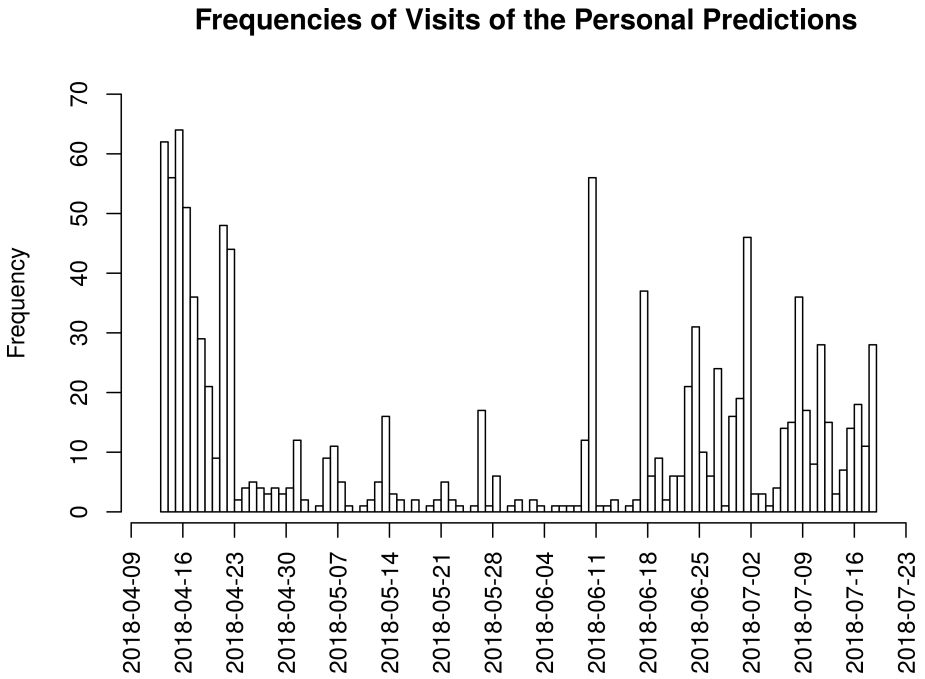


Figure 4.9: Frequencies of visits of the personal skipping and examination fitness predictions, throughout the courses duration. Notable peaks of activity are visible for the start and the end of the course.

for both the skipping and the examination fitness predictions whether these predictions motivated them to learn more, whether they were interesting, discouraging, helpful, or motivated them to hand in the next assignments on a six-point Likert scale ranging from "not at all" to "absolutely". The results can be seen in Figure 4.10. In general, it can be concluded that students did find the displayed predictions moderately interesting, yet not particularly encouraging or discouraging (while the skipping predictor seems to be perceived as a bit more discouraging, than the examination fitness predictor).

Attitudes towards analytics reports. A second case study was conducted in a course on logic and discrete mathematics in 2019 (LDM-2019 described in Appendix A). For this case study, the software was extended to send analytics reports and to indicate changes of predictions along with the personal predictions. About 10% of the students chose to disable their analytics reports, with none of them handing in a single assignment.

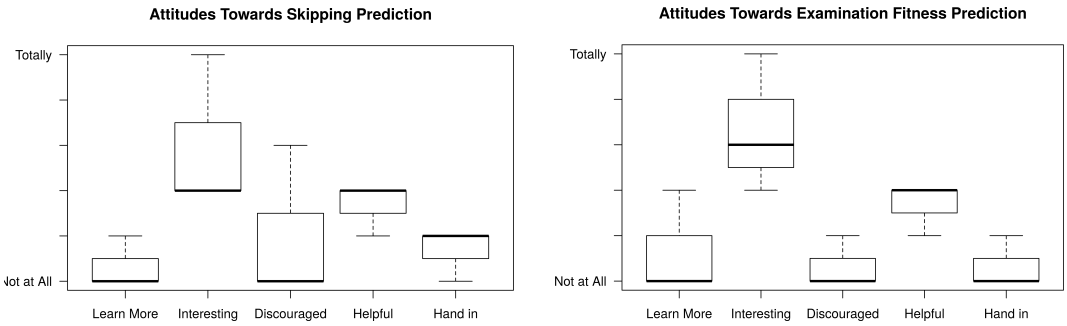


Figure 4.10: Attitudes towards predictions. Left: Skipping predictions. Right: Examination fitness predictions.

In a final survey, students indicated whether the reports encouraged or discouraged them, or if they were indifferent towards the reports. Two-thirds of the students indicated indifference, with the rest of the students equally often indicating encouragement and discouragement. In a open item question, students stated general remarks towards the reports, which one student used to indicate his discouragement: "After the first 6 weeks of the semester, I had a score only 2 per cent better than my friends, even though I had submitted every exercise so far and my friends none. It was very discouraging to see that doing these exercises didn't improve my chance of passing the exam." While this observation could not be reproduced by testing the software, even *perceiving* such a software behaviour is undoubtedly discouraging.

Students were also asked to indicate reasons for not submitting homework. The given reasons included a lack of incentives like bonus points, specifically focussing on homework for other courses that provide incentives, and doing the homework but not uploading them because no feedback was deemed necessary. Also, technical problems and a confusing user interface were mentioned several times. Interestingly, students did not complain about problems with the same user interface in the case study conducted in TCS-2018 (where bonus points were awarded for delivering homework).

Discussion and perspectives. All in all, the results regarding the student attitudes towards analytics-based nudging are sobering, as they range from annoyance to ignorance with a minority of students being encouraged or motivated by the software. On the other hand, students found the provided analytics interesting, a finding which is underpinned by statistics illustrating that students frequently retrieved their personal predictions.

Arnold et al. report higher retention and generally positive attitudes of students being nudged with e-mail reports similar to those described above [8]. Yet, many students perceived the e-mails generated by the system described by Arnold et al. as personal communication with their instructor – which it was not (initially) as they were completely computer-generated. While it can be argued that students would feel more encouraged by analytics reports feigning human correspondence, this approach was well knowingly not chosen for Backstage 2 / Projects. It is the author's conviction that students should know if they were nudged by a person or by software. An interesting perspective on that problem is provided by the OnTask system, which lets teachers compose *message templates* specifying which text passages are to be included under which conditions (e.g. including nudging passages on if the student did not yet access certain learning materials) [183]. In this way, teachers could *choose* inform learners about this analytics-driven message personalization.

Some students reported problems with the software as reasons for not submitting homework. Remarkably, these comments were only made by participants of the course where no bonus points for homework were granted. In the courses with bonus points, no such problems were reported in the final surveys, while, in a handful of occasions, students decided to contact the teaching staff during the course and asked how they could submit homework. This observation is important when considering a further evaluation of the software (or similar software such as online homework systems): Maybe the bonus points incentivised students to accustom themselves to an unfamiliar software, where the non-incentivised students could not be bothered to do so. However, students reporting not submitting homework due to software inconveniences may suffer from a self-serving bias [162] and systematically overemphasise external reasons for skipping homework (unhandy software) while underemphasizing internal reasons (such as lack of motivation).

4.4 Peer Teaching

This teaching format results from the obligation at the author's faculty to conduct examinations for bachelor degree computer science courses twice a year and in each semester. While this regulation is sensible as it prevents students from having to wait long periods to take an examination (possibly completing their course of study), this regulation also poses organizational problems for the teaching staff who can offer courses only once per year. The teaching format described in the following tries to alleviate this problem by requiring minimal teacher intervention and relying on orchestrated peer teaching.

Peer teaching is defined by Goldschmid as any form of instruction where learners teach each other [95]. Among the benefits of peer teaching are improved teamwork abilities and social skills among learners [225] and increased comprehension of the subject matter for both student teachers and learners [20, 21].

The format described in this section orchestrates *peer review*, which is a form of *peer teaching* defined as a "reciprocal process whereby students produce feedback reviews on the work of peers and receive feedback reviews from peers on their own work"[172]. The efficiency of peer review for learning has been established in several meta-analyses [69, 83, 241], and, similar to peer teaching in general, positive effects of both *reviewing* other learners and *being reviewed* by other learners have been reported in the literature [42, 172, 153]. Note that, in the literature, peer review is often discussed as part of a teaching format which also encompasses face-to-face meetings with the teacher. In the format described in this section, this is not the case: No face-to-face teaching was conducted in this course.

The results presented in this section were initially published in [108], an extension of which can be found in [107].

Format definition. As in the teaching format discussed in the last sections, subprojects of a general course project are used to organize material and exercises on different topics, where a new topic is introduced every week. In this way, students are provided each week with lecture material covering a topic and homework assignments requiring the application of the newly acquired knowledge. After having handed in their homework, each student is tasked to review submissions of two other students. Students have a week each to deliver homework and to deliver peer reviews, which leads to an interleaved schedule:

1. At the beginning of the first week of a topic, course material and corresponding homework assignments on the topic are published. Students are tasked to deliver homework within that week.
2. At the beginning of the second week of a topic, each student is tasked to review the homework of two other students. They are tasked to deliver their reviews within that week.
3. At the beginning of a topic's third week, exemplary solutions for the homework assignments are published. Students are tasked to correct their assignments by using the received reviews and exemplary solutions. Note that a self-correction of homework in that manner is regarded to have beneficial effects on learning [203]. No deadline is given for performing self-correction.

Note that this schedule realizes a form of spaced instruction (requiring students to reconsider a topic regularly), which is known to be beneficial for learning [227].

To realize this schedule, an adapted form of the homework-delivery-review script given in Listing 4.1 was used, choosing students instead of teachers as reviewers and generating two (instead of one) review assignment for each completed homework. Student participation

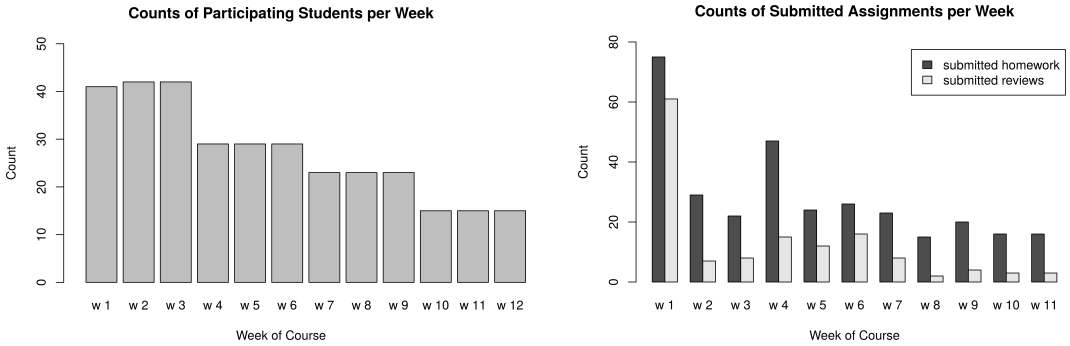


Figure 4.11: Participation development throughout the course. Left: Development of the number of participating students. Right: Comparison of delivered homework assignments and delivered reviews. Figure adapted from [107]

both in delivering homework and in providing reviews is crucial for the operation of this teaching format, as the goal is to enable learning by both giving and receiving feedback. To ensure participation, reminder emails are sent after two consecutive missed assignments (not regarding whether homework or review assignments were missed), and students are excluded from the course after having missed three consecutive assignments.

Results: Participation and student competence. The teaching format was used in an introductory computer science course on functional programming using the programming language Haskell (see FP-2017-p in Appendix A). The course lasted 13 weeks from October 2017 to January 2018 and counted 45 students. In total, 27 homework assignments covering 11 topics could be worked out by the students according to the review scheme described above. Topics were covered by two or three exercises each.

Throughout the course, student participation declined, and students were excluded from the course as a consequence of the exclusion rule. Figure 4.11 illustrates this development. Note that far fewer review assignments were completed than homework assignments (while each delivered homework led to the creation of two review assignments). In total 316 homework submissions and 147 reviews were delivered, which means that less than half of all delivered homework was reviewed. The distribution of reviews given in the course followed a long tail distribution, with 18 of the 45 students giving 90% of the reviews. A more detailed description of student participation is given in [107].

Students were asked in a final survey whether they dropped out of the course, and if so, why. Of the 18 responses gathered in the survey, the most prevalent reasons for dropout were time constraints and other courses, and lack of motivation due to missing feedback.

Both the homework and reviews delivered by the students varied largely in quality. To assess the review quality, two teachers categorized all reviews given in the course after the course's end. A review is hereby counted as "correct" if it either correctly points out errors in a submission or correctly assesses its correctness. Accordingly, a review is "incorrect" if it fails to point out errors, or points out errors that are not present in the reviewed submission. This definition led to the following categorization:

- **+FF** : "false correctly reported by the reviewer as false", accounting for 25% of the reviews.
- **-FC** : "false wrongly reported by the reviewer as correct", 22% of the reviews
- **+CC** :"correct correctly reported by the reviewer as correct", 47% of the reviews
- **-CF** : "correct wrongly reported by the reviewer as false", 6% of the reviews.

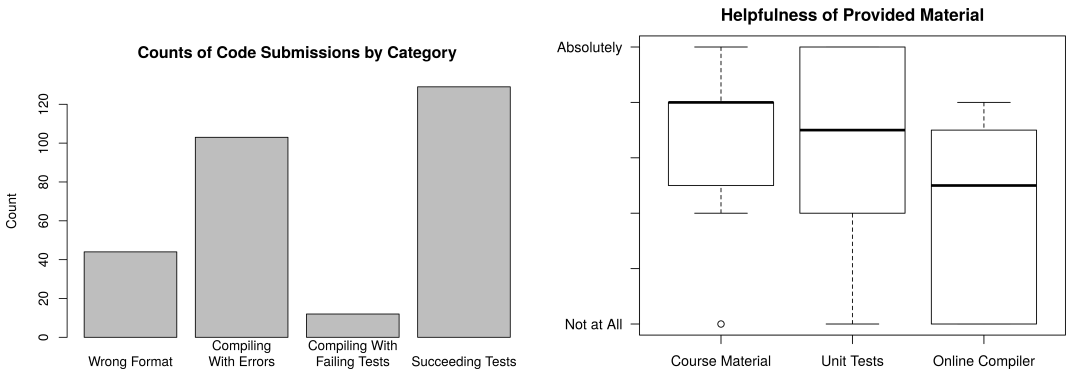


Figure 4.12: Left: Absolute numbers of code submissions in the respecting categories. Right: Perceived helpfulness of provided material, unit tests, and online compilation functionalities.

Frequencies of +FF and +CC significantly correlated positively (Pearson's $r = 0.44, p < 0.05$), and frequencies of +FF and -FC significantly correlated negatively (Pearson's $r = -0.45$, $p < 0.05$). In other words: Students who were able to correctly identify errors were also able to correctly identify the correctness, as well as being less susceptible to making assessment errors. This motivates the definition of a student's *review quality score*: The difference of a student's relative frequency of correct assessments (+FF and +CC) and incorrect assessments (-FC and -CF). To assess the homework quality, an automated procedure relying on the compilation service of the software (see Section 4.1) was used.

Homework submissions were automatically categorized according to the following scheme:

- "wrong format": For submissions that contained no code (such as PDF files), encompassing 15% of the submissions.
- "compiling with errors": For submission that could not be compiled (e.g., due to syntax errors), 37% of the submissions.
- "failed unit tests": For submissions that could be compiled and executed, but failed at least one of the provided unit tests, 6% of the submissions
- "succeeding tests": For submissions succeeding all tests, 42% of the submissions.

This categorization is only sensible for programming homework, which constituted 23 of the 27 assignments. The quality of the remaining 5 non-programming assignments (usually requiring some kind of argumentation) was not assessed. Figure 4.12 illustrates the category sizes.

The students' *review quality scores* correlated significantly with review participation ($r = 0.4$, $p < 0.05$): Students giving the best reviews (in the sense of correctness), gave the most reviews. Review participation, meanwhile, correlated significantly with coding proficiency, calculated as the relative frequency of code submissions compiling without error (Pearson's $r = 0.35$, $p < 0.01$). In summary: Proficient students gave more reviews which were, on average, of higher quality, than their less proficient peers.

Comparing examination results with student behaviour yielded mixed results: Students participating in the course were not more successful in the final examination than students preparing otherwise. Furthermore, examination success of course participants did not correlate with the number of submitted homework, as it did in the "traditionally" conducted course preceding this course. Figure 4.13 illustrates the difference in the relation of counts of delivered homework on examination success for both courses: In the "traditional" course there are few students delivering homework and having low grades, while many students

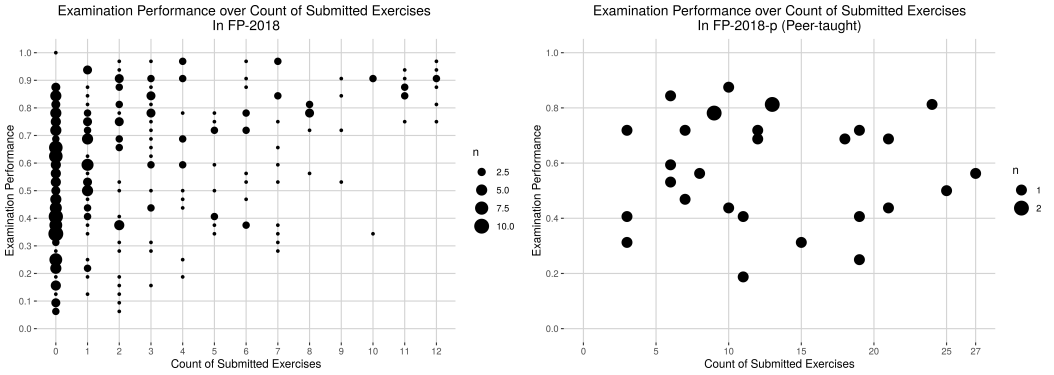


Figure 4.13: Comparison of the relation of count of submitted exercises and examination performance. Left: "traditional" relying on the teaching staff. Right: Peer teaching. Note that exercises on the left are aggregated by week.

are delivering no homework and having high grades. In this "traditional" course, delivering homework *resulted* in good grades, while good grades could also be achieved by students delivering little or no homework. In the course relying solely on peer teaching, no such implication can be seen. Yet, examination success in the peer taught course correlated with the relative frequency of *received reviews*. It seems that feedback is the main contributor for homework efficiency: In the preceding course, all homework was revised by the teaching staff, while in the peer taught course, feedback was often missing, possibly resulting in students generally not benefiting from their delivered homework. Not surprisingly, coding proficiency correlated positively with student success in the peer taught course.

Results: Attitudes. In the final survey, students indicated that the provided material on the platform and the provided unit tests accompanying the exercises were helpful (medians 5 and 4.5 on a 6-point Likert scale ranging from "not helpful at all" to "absolutely"). The provided online compilation functionalities were perceived as less helpful (median 3.5 on the same scale). Figure 4.12 illustrates these findings.

Notable are differences of perceived properties of *given* versus *received* peer reviews: Most students indicated that the peer reviews they received were only sometimes helpful, and only sometimes identified errors correctly (see Figure 4.14 left). On the other hand, students were relatively confident that the reviews they *gave* were helpful for others (responses exhibited a median of 4 on a 6 point Likert scale ranging from "not helpful at all" to "absolutely helpful").

Most of the students indicated that the process *of giving reviews* was helpful for their learning, gave them new ideas and allowed them to compare their work with other solutions (see Figure 4.14).

Discussion and perspective. The main problem of the peer teaching format identified by the evaluation is low participation, resulting in few delivered peer reviews, which in turn was reported to decrease motivation to participate. Further developments of this teaching format would need to address this problem. Yet, the teaching format was already designed with this problem in mind: Not participating led to the exclusion of students from the course, and each homework was reviewed by two peers, a measure taken among others to compensate for missing reviews. One can argue that the exclusion rule was not strict enough and that reviewing could have been assigned to more than two students in parallel.

The 15 students who participated throughout the course (and did not drop out) skipped about half of their assigned reviews (51%). Hence, if the course had started with only these

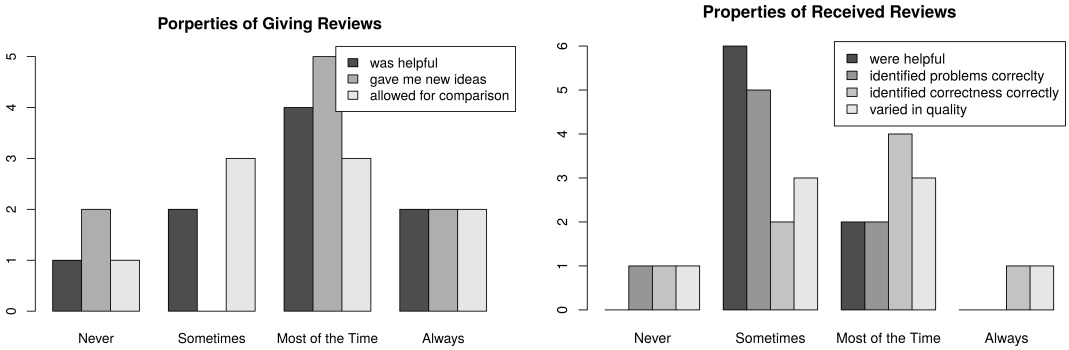


Figure 4.14: Left: Beneficial properties of giving reviews. Right: Properties of the received peer reviews.

15 (or if the others would have been excluded very early from the course), the policy of letting two students review the same submission for redundancy could have worked out, possibly providing a sufficiently large coverage of feedback to sustain participation.

Another measure for improvement would address *feedback efficiency* instead of *feedback coverage*. The evaluation revealed that more proficient students gave more and better reviews. By letting more proficient students review the less proficient and vice versa, feedback could be more efficient: The less proficient would profit from more and better feedback, while the more proficient would (hopefully) not be hindered too much by missing feedback. As the proficiency was estimated automatically (by searching source code for errors, compilation messages, and compliance to unit tests), this could be implemented without increasing the workload of the teaching staff.

Interestingly, there are no reports in the literature of peer review quality being influenced by the reviewer's competence. The author proposes three explanations for this unusual result: Firstly, the dataset available for evaluation is small, relying on only 45 students, and this effect (while being significant to the 5% threshold) might result from noise. Secondly, there are many interpretations of the term "peer review", and the results may depend largely on details of the implementation. Many authors include peer review in teaching formats including face-to-face teaching sessions and others implement non-vocational peer review [191]. Indeed, Pirttinen et al. reported that only the best students gave peer reviews in a course on programming [191], which is a finding comparable to those presented above. Thirdly, much of the literature on peer review is concerned with reviews on essays and homework written in natural language (instead of programming homework reviewed in this evaluation). Yet, there is a fundamental difference in reviewing, for instance, an essay *explaining* the principle of recursion, and reviewing a program *using recursion*: When reviewing an essay, the reviewer can assess (to a certain degree) its quality even without a sufficient understanding of the topic. If the reviewer does not understand recursion, and the essay does not convey that understanding (while it should!), they can assess the essay as insufficiently comprehensible. When assessing the quality of a recursive program, on the other hand, the reviewer needs to understand recursion, simply to assess whether the reviewed program is indeed recursive (let alone assessing qualities such as termination, correctness, and technique).

If this third explanation is (at least in part) correct, then orchestrating peer review needs to be adapted in courses which heavily rely on formal languages or mathematical formalisms, using the more proficient students (giving more higher-quality reviews more reliably) to revise the work of the less proficient students. Also, further experimentation with the teaching format could try to estimate a "tipping point" of participant composition: What

distribution of competence among the course participants is necessary to make the peer teaching format sustainable?

To assess "proficiency" among the students, automated code categorization was used. These results led to the implementation of a novel predictor, using automated code assessments to predict "levels of competence" (for instance the ability to provide semantically correct code). This predictor is described and evaluated in Section 5.4. A second case study, relying on the same student-driven teaching format, was conducted in a course on discrete mathematics. For this second case study, the orchestration script presented earlier was edited not to choose review-reviewee pairs at random but to maximise the difference in their estimated competence levels. Yet, the evaluation yielded no conclusive data due to the lack of student involvement (which was lower from the beginning than in the case study presented above).

As a last remark, consider the social dimension of this course format which orchestrates *peer review* as a starter for *peer teaching*. In the evaluated course, reviews were rarely commented on by the reviewed students, which would indicate peer teaching in the form of a discussion. Further improvements of this teaching format could involve emphasizing social aspects of the software.

4.5 Exploratory Learning of Formal Languanges

Exploratory Learning is a pedagogical approach in which a learner "investigates a system on their own initiative, often in pursuit of a real or artificial task" [210, p. 1]. Exploratory learning found its first applications in the early 1980s in the field of software training, where it was found that users could, and were willing to, learn to operate software applications through exploration and discovery, and without formal training [39]. The related term Discovery Learning is more common in the formal education literature and refers to "a type of learning where learners construct their own knowledge by experimenting with a domain and inferring rules from the results of these experiments" [245, p. 386]. Note that both terms (Discovery Learning and Exploratory Learning) are sometimes used interchangeably [173]. Software is commonly used to realise exploratory learning by providing the learner with a visualization of the conceptual space to be explored and tools for exploration, a so-called microworld [182]. Microworlds have found numerous applications in different STEM fields, examples of which are discussed in greater detail in Section 3.2.

Backstage 2 / Projects' functionalities which aim to sustain the exploratory learning of STEM's formal languages are the code editor Knoala and the compilation and execution service Coconut described in Section 4.1. As both components cannot be used separately by the platform's users, "using the compile and edit functionalities" is just referred to as "using Knoala" in the following.

Knoala was used to sustain learning in all courses evaluated for this research. Learning the theory of computer science was supported with simulators of Turing Machines and push down automatons, evaluators for theoretical programming languages WHILE, LOOP and GOTO, and functionalities to evaluate expressions of primitive recursion and μ-recursion. Courses on logic and discrete structures were supported with functionalities to evaluate expressions in predicate logic, with a scaffold easing the transformation of logic expressions to clause normal form, and with a scaffold verifying proofs by structural induction. Courses were also supported with "practical" programming languages such a Haskell compiler for courses on functional programming.

Theoretical programming languages, simulators, and mathematical formalisms are like "practical" programming languages (like Python or Java), as they follow a non-ambiguous

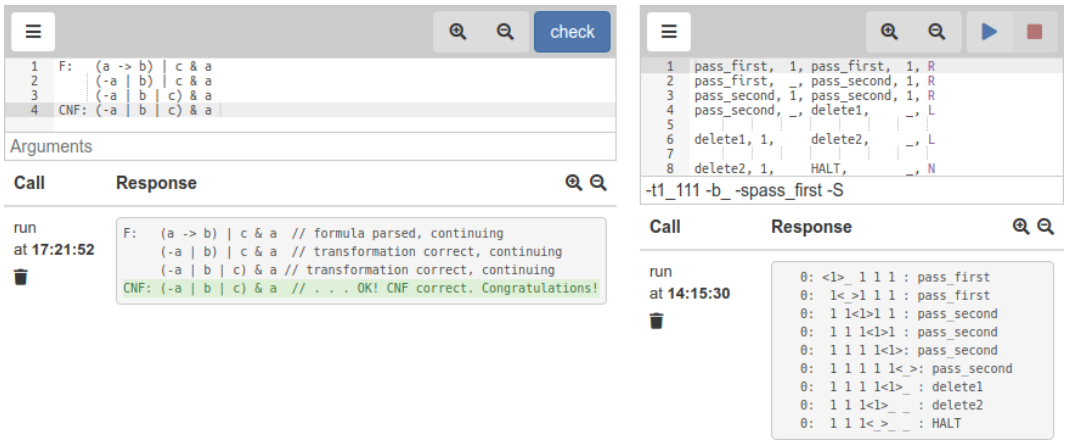


Figure 4.15: Different interactions for different formal languages. Left: "checking" a transformation of a logical expression into clause normal form. Right: "running" a Turing Machine simulator.

syntax which can be verified by software. Yet, these formalisms are used to prove mathematical theorems, not to develop real-life applications: The programming language LOOP, for instance, is used to explore which functions can be expressed by fixed-length loops. Note that theoretical programming languages are "executed" or "run", while mathematical formalisms are "verified" by Knoala. This difference is represented by providing different interactions available for different languages: The interactions of the Turing Machine simulator are "run" and "stop" (See figure 4.15 right), the interaction provided for a transformation of a logical expression into clause normal form is "check" (See figure 4.15 left).

Knoala was commonly used for delivering and assigning homework: If students were tasked to provide homework requiring the use of a formal language, they could use Knoala to test their solution. Also, the exercise text provided by the teachers sometimes contained a partial solution for the students to complete. Further, worked examples and additional material were often provided using Knoala. Even descriptions of systematic errors (see Section 4.2) could be provided using a formal language (giving an example of the error), yet this was not done in any course by the teaching staff.

While the support for formal languages is undoubtedly a versatile learning and teaching tool, it is not a teaching format on its own. Yet, it is a ubiquitous part of the software which was repeatedly used to accompany whole courses (such as the analytics-based nudging presented in Section 4.3), and therefore student attitudes toward these functionalities and several striking patterns of behaviour in its use are reported in this section. The presented results are a summary of the results published in [105].

Software characterization. To the author's best knowledge, "exploratory learning for formal languages" is rarely discussed in the literature, or if so, only implicitly: Students learn formal languages by specifying operation codes for simulated machines [251], or mini-languages that steer simulated robots [32]. On the other hand, several studies explore tools which scaffold the learning of mathematical formalisms, where the focus typically lies on solving exercises, measured by a correct input: Entering a correct formula [124], a correct numerical value [187, 91], or a correct single line of code [6]. These forms of exercises have been shown to encourage trial and error behaviour, where learners try to produce a correct solution by trying out different possibilities instead of reconsidering the problem [187]. Also, this view of "using a formalism to solve an exercise" is quite narrow, because mathematical formalisms can be used to generate new ideas through exploration, as argued by Dubinsky: "it is not only possible, but a standard activity of mathematicians, to use

formalism to construct meaning and this can also be a source of mathematical ideas" [73, p.1].

Consider for that matter the scaffold supporting the transformation of expressions in predicate logic depicted in Figure 4.15. It can be regarded as an explorable microworld: There are few very restrictive rules for manipulation (distributivity, commutativity, etc), the tool is easy to use (if the user knows how to use a text editor), and it contains a minimal set of objects that are subject of the exploration (the logical variables, and the expressions). Also, this scaffold does not have to be used to construct clause normal forms, in fact, it just checks whether all two consecutive lines contain equivalent logical expressions – it just displays a special message if the last line is in clause normal form.

While many of the aforementioned tools use graphical user interfaces to represent parts of the artefact produced by the learner as interactive elements (such as blocks or buttons)[33, 93, 233], this work focusses on text-based interfaces for several reasons:

- *simplicity*: Text documents can be easily exported from and imported to the software without requiring a special encoding. Also, new documents containing, for instance, an exercise description can be composed with the same software as the learner would use to solve the exercise.
- *commenting*: It is common for programmers to interleave formal language (the source code) with natural language (comments, which are not part of the formalism). Undoubtedly, commenting is useful for many educational purposes: Stating hints or problems, leaving remarks, or simply to exclude parts of the work from interpretation without having to delete them. Text-based formalisms usually support this functionality by introducing symbols (which are not part of the formalism) that mark all following symbols of that line as comment text, which an easy to use approach.
- *extensibility*: To support a new formalism, large parts of the same software (providing the text editing functionalities) can be re-used. Note that implementing the "language-specific" parts of the software can be achieved with techniques from programming language design (relying for instance on parsers and parser generators and frameworks for syntax highlighting).

Evaluated data. The evaluation presented in the following reports solely on the use of theoretical programming languages and mathematical formalisms, not on"practical" programming languages. This restriction was made for two reasons: Firstly, in this way, most (if not all) of the students' interactions with the provided languages could be observed. Most students, especially the more proficient, would arguably prefer to edit, compile and run programs in "practical" programming languages on their computers instead of relying on a web application. Yet, the theoretical programming languages and formalisms evaluated in the following were only available through Backstage 2 / Projects. Secondly, this restriction ensures to observe many students' *first* interactions with the examined languages. Many students have previous experience with "practical" programming languages which could bias the results.

The data presented in the following was gathered in two courses: A course on theoretical computer science lasting from April 2018 to July 2019[10] attended by 433 students (of which 85 used Knoala at least once), and a course on logic and discrete mathematics[11] lasting from September 2018 to February 2019 attended by 42 students (of which 17 used Knoala at least once). Throughout these courses, several formal languages were introduced, with the following homework assignments requiring their use.

[10] TCS-2018, see Appendix A
[11] LDM-2018, see Appendix A

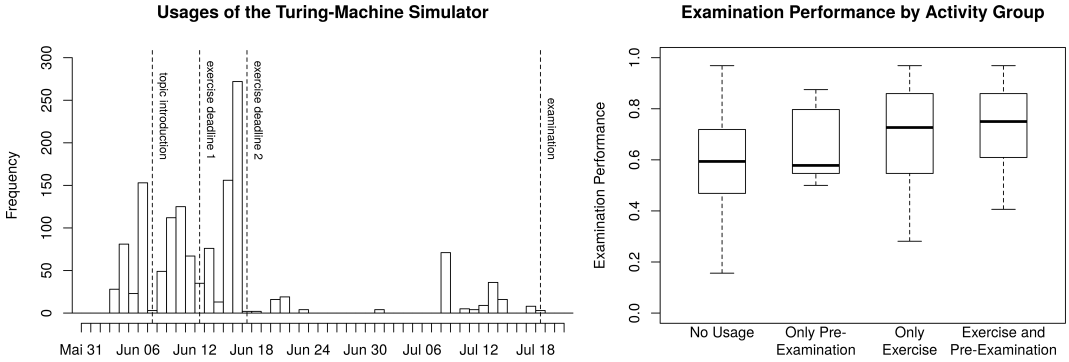


Figure 4.16: Left: Activities of the Turing-machine simulator from introduction of the topic, two homework deadlines to the final examination. Right: Examination success students, grouped by types of activity.

Results: Patterns of use. Two phases of activities could be distinguished: An "exercise phase" shortly before or after a formal language was introduced and homework had to be delivered, and a "pre-examination phase" shortly before the final examination (see Figure 4.16 left). Students using Knoala in both phases or only in the first phase were significantly more successful in the final examination ($p < 0.05$) than students using Knoala only shortly before the examination or not at all (see Figure 4.16 right).

Similar to other activity counts reported in the previous sections, Knoala's usage was very unequally distributed: 35% of the students performed 90% of the absolute activity (i.e. executed or automatically verified code), with only 39% of the students using Knoala more than twice. These 39% of all students are referred to as "fiddling" students, as their behaviour entailed a more intensive use and more engagement with Knoala when compared to the behaviour of their "non-fiddling" peers:

- Fiddling students made significantly more use of the examples and templates provided by the teaching staff than non-fiddling students.
- Fiddling students made syntax errors with a significantly lower frequency, and syntax errors happened later in their active phases. In turn, these active phases were significantly longer.
- Non-fiddling students, on the other hand, edited more between tests and re-tested the same code without modifying it more often. All code which was re-tested unchanged contained syntax errors.

Results: Attitudes. In August 2018, a survey was conducted among the participants of the course on theoretical computer science, evaluating Knoala's perceived helpfulness and ease of use. Students were asked on a six-point Likert scale ranging from not at all (1) to absolutely (6) to indicate Knoala's helpfulness for learning, its ease of use, and whether they found running code of other students easy. 27 students participated in the survey. The results are depicted in Figure 4.17: Knoala was perceived as helpful (response median of 5), not so easy to use (response median of 3), and compiling code of other users was perceived as not easy (response median of 2.5).

Also, students indicated in a general open item question the functionalities they liked best of Backstage 2 / Projects. 13 students (about half of all participants) mentioned Knoala in positive statements like "The language compiler/ interpreter is a really cool feature" or "The online compilation tools are somewhat useful.", while a request for better error messages was mentioned several times.

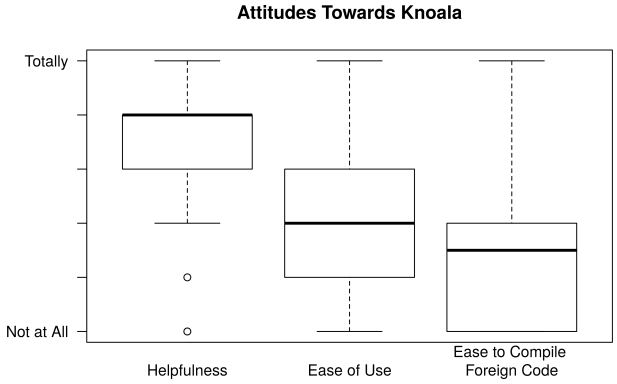


Figure 4.17: Survey results regarding the helpfulness, the ease of use and the ease to test foreign code.

Discussion and perspective. Several observations suggest that students benefited from Knoala: Many students used the tool to deliver homework and to prepare for the examination, and some even explored formal languages before their introduction in class. Students using the tool were more successful in the final examination, and students, in general, found the tool helpful for learning.

It is interesting to compare the perceived helpfulness of "practical" programming languages and "theoretical" programming languages and formalisms. In a course on functional programming with Haskell[12] (reported about in Section 4.4 and in [108] and [107]), students found the online compiler much less helpful than in theoretical computer science (response mean of 3.5 for functional programming, 4.4 for theoretical computer science on a scale ranging from not at all (1) to absolutely (6)). This may be because many of the theoretical formalisms were available exclusively through Knoala, while the Haskell compiler is freely available.

For teachers providing feedback, Knoala's benefit is obvious: Assessing whether a program indeed does what it should is much easier if one can run the program, and the similar statements can be made for proofs, or expressions in any formal language. This advantage of online homework systems supporting mathematical formalisms over hand-written homework submissions is often mentioned in the literature [124, 34].

The goal of this teaching format is to encourage exploratory learning of STEM's formal languages. Unfortunately, the data presented here cannot establish for certain whether students used the Knoala in that way. Exploratory learning could be detected for instance if students would evaluate expressions of a formal language which not directly solves one of the provided exercises. From the currently available data however, it cannot be deduced *for which purpose* Knoala was used, not even whether an active phase with Knoala led to the completion of homework. Yet, certain behaviours of "fiddling students" (the minority of students performing the majority of activities) seem to relate to exploratory learning: They regularly used templates and examples provided by the teaching staff to start their active phases and worked in longer phases than their non-fiddling peers.

The evaluation revealed that the error messages provided by Knoala should be improved: This is indicated both in the survey responses and by the fact that non-fiddling students made many syntax errors which happened early in their active phases. Possibly, better error reporting could have helped these students. It has to be noted that implementing "good" error messages is subtle, as this anecdotal experience illustrates: One student submitted a homework in the course on theoretical computer science in 2018 containing a WHILE-

[12] FP-2017-p see Appendix A

program with an (obviously frustrated) remark stating that "nothing works on this site" (referring to Backstage 2 / Projects and Knoala). The submitted program contained an infinite loop, leading the compiler to report (correctly!) a timeout error after 20 seconds of computation. Learning that WHILE programs can get trapped in infinite loops (while other computational models discussed in the course cannot) is an important learning goal, and this student could have learned this fact in that instant, yet the displayed error message was misunderstood as an error of Knoala instead of the evaluated program.

A laboratory study was conducted to establish whether better error messages could in fact "lower the syntax barrier", and the positive results of this study are presented in Section 6.3. Yet, the syntax barrier could also be lowered without additional software. "Fiddling" students used the examples and templates provided by the teaching staff more often, and, if so, always from the beginning of an active phase. Possibly, these students took advantage of the syntactically correct examples by modifying them and "fiddling" until the solutions fit their needs. This could be integrated into the teaching practice with assignments asking the students to modify or complete syntactically correct programs, instead of letting them produce complete programs from the start. Also, learning analytics could be used to determine *how likely it is* that a student will be able to produce syntactically correct code, and the choice of assignment could be adapted accordingly. An approach for predicting such student qualities is presented in Section 5.4.

The observed behaviour of "retesting code without changing it", typical for non-fiddlers, is to some degree irrational: Incorrect code will not start working on its own. Therefore, this behaviour can be interpreted as a sign of frustration. In the future, Knoala could detect such behaviour and establish contact between the possibly frustrated student and a tutor or peer.

CHAPTER 5

Predictive Learning Analytics

In this chapter, four predictors of learning behaviour are introduced, which predict skipping and absenteeism (Section 5.1), examination performance (Section 5.2), systematic errors and misconceptions (Section 5.3), and levels of programming competence (Section 5.4). The skipping and the examination performance predictors are *task-agnostic*: They predict a behaviour independent of the task a student attempts next (for instance, every task can be skipped). The misconception and the competence level predictors are *task-specific*: They make predictions which pertain to a specific task. For instance, a misconception related to the use of arithmetic expressions is most likely to occur in exercises relying on arithmetic.

Backstage 2 / Projects supports each of these predictors, which can be added to any project on the platform. After a predictor is added to a project, it has to be trained with data previously gathered on the platform. Once trained, the predictor will make predictions based on data gathered in the project as described in Section 4.1.

While the accuracy of each predictor was evaluated with data gathered on the platform, only two of the predictors were used to accompany running courses: The skipping predictor described in Section 5.1 and the examination performance predictor described in Section 5.2 were used to implement *analytics-based nudging* as described in Section 4.3.

The sections in this chapter are organized as follows: Each section introduces a predictor by describing the statistical methods used for prediction, how the predictor is trained, and how predictions are computed. Further, accuracy measures are reported and perspectives for its future use and improvements are developed.

Data sources. To compute learning analytics, a multitude of data are considered in the literature such as login times and activity measures [1, 52, 78], demographic data [52], and performances in previous courses [52, 98].

Arguably, much of the data collected in these ways, such as login times or counts of course material visits, may reflect passive learning. The predictors presented in the following use data on *active learning*: Artefacts generated by learners during or as a result of their learning such as homework submissions, delivered source code, quiz results, etc. An often used data source are quality assessments of homework submissions, which were either

performed by the teachers of a course (Section 5.1, 5.2, and 5.3), or automatically (Section 5.4). Using teacher assessments as a data source seemed to be particularly powerful for behaviour prediction, possibly because humans can detect problems that are difficult to automatically assess: A solution for a programming assignment may be technically correct (which can be detected automatically), but it might be overcomplicated (which is difficult to automatically assess). Yet, students might have problems later if such a programming style was maintained; problems which could be predicted *because of* the teacher assessment.

The predictors aim to improve learning by predicting (possibly problematic) learning behaviour *within a course*. Therefore, all predictions are based solely on data gathered *within* the considered course. While "course-independent" data (such as previous performances and demographic data) have been successfully used in the literature to predict learner behaviour [52, 98], such data was not considered as it would rather serve to *cluster* students beforehand instead of predicting learning problems based on observed behaviour.

Predictor requirements. As all predictors only use "course-dependent" behaviour data, no data about a student's recent behaviour is available at the beginning of a course. The predictors are designed to make predictions *from day one* of a course, with predictions usually increasing in accuracy as more data is gathered. To measure the quality of a predictor, average performances are reported in the following.

A challenge in designing behaviour predictors based on educational data is to achieve robustness against data irregularities: Students may join a course several weeks late, different course venues may treat the same topics but in a different order, and course lengths may differ between course venues. Indeed, in all the datasets evaluated for this research, no two course venues were conducted *completely* identically. The predictors introduced in this section are designed with this problem in mind, and further considerations on this issue can be found in the sections introducing the specific predictors.

Predictors for nudging. One application of learning analytics is to "nudge" students towards a better learning. This can be achieved by predicting problematic learning behaviour and providing automated personalized interventions accordingly. Interestingly, these interventions aim to falsify the prediction made: To avoid problematic behaviour. For instance, if certain systematic errors were predicted for a learner, and they would indeed *not make* these errors (due to the prediction being reported to the learner), the predictor would make a false positive prediction. Hence, by using a predictor for nudging, one would expect *to reduce its accuracy*. It is an open question how data gathered in a course encompassing predictor-based nudging could be used in the future: If specific problems did not occur (or occurred less often) due to such a nudging, a predictor trained on this freshly gathered data would most certainly fail to predict exactly these problems.

The author proposes three desirable properties for behaviour predictors for nudging, which have guided the design of Backstage 2 / Projects' predictors:

- **Intelligibility:** It should be clear on which data a prediction is based, and how this data reflects the student's learning. Predicting examination performance with performances in previous courses (as performed in [98]) is not intelligible in this sense, as it is usually not clear how the learning of topic A reflects the learning of topic B.
- **Actionability:** Predictions should reflect a students changes in behaviour. For instance, if the software warns of a problematic behaviour such as skipping homework, and the student does indeed avoid that behaviour (and delivers homework), further predictions should reflect that change. Note that intelligibility can be of greater importance than actionability: Elbadrawy et al. used (among others) simple forum activity to predict course performances [78]. Such a prediction is certainly *actionable* as it can be improved by being more active in an online forum, yet it is not *intelligible* as not every kind of forum activity relates to learning.

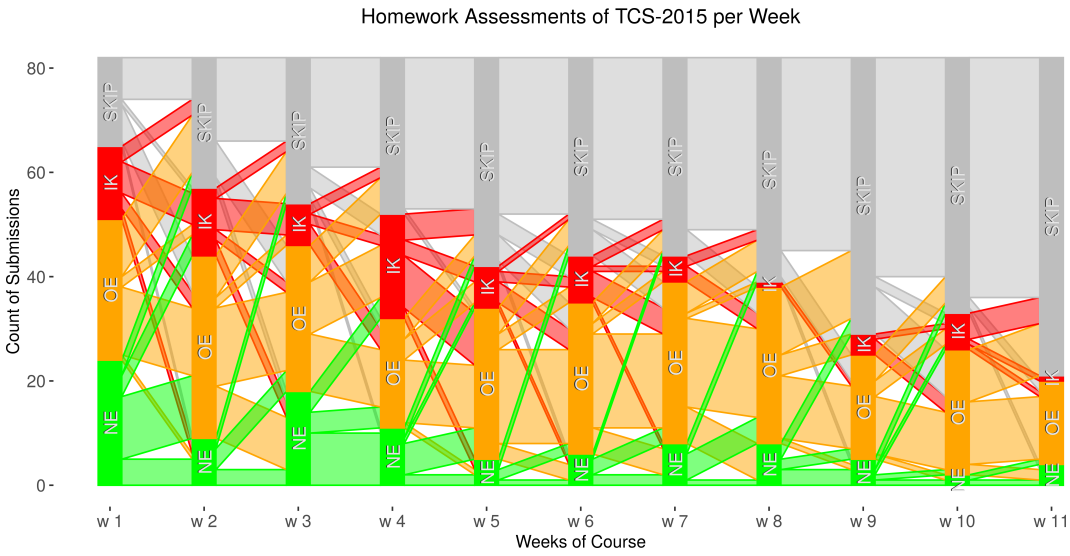


Figure 5.1: Homework assessment for weekly homework assignments in a course on theoretical computer science (TCS-2015), heights of coloured bars indicate counts of submissions in the respective categories. Beams connecting categories in subsequent assignments indicate counts of students subsequently submitting homework in these categories.

- **Accuracy:** Obviously, predictions should be accurate. Yet, actionability can be of greater importance than accuracy: A predictor using a student's data (such as gender and race, as done in [52]) might be more accurate than a predictor not incorporating that data. Yet, these predictions would only partially reflect a student's actions, hence they would not be actionable and therefore not be suited for nudging.

5.1 Predicting Skipping and Absenteeism

The predictor introduced in this section estimates the probability of a student to skip his or her next homework. It aims to identify students *on the verge of skipping*, to allow a sensible intervention. Figure 5.1 shows the development of skipping behaviour of students throughout a course of theoretical computer science.[1] Homework submissions were categorized based on the scheme introduced in Section 4.3:

- **NE** for "no error"
- **IK** for "insufficient knowledge"
- **OE** for 'other error"
- **SKIP** for skipped activities

Two observations can be drawn from the data: Firstly, students who skip homework once *rarely* rarely submit homework afterwards. Secondly, students showing *insufficient knowledge* in one assignment are more likely to skip than students making no or other errors. Hence, a possible cause of skipping may be lagging behind and finally losing track of the course.

Similar observations have been drawn in the literature: Robins explained bimodal grade distributions[2] by simulating learning behaviour [211]. Each student was simulated to have

[1] TCS-2015 see Apendix A.

[2] Bimodal grade distribution: Outcomes with both a high number of very well performing and a high number

a certain "momentum", signifying their grade of activity, which could be lost during the course (for instance by failing to learn a topic). Similarly, Kizilcec et al. analysed patterns of learner behaviour related to disengagement in MOOCs by automatically categorizing the participants' behaviour as either "Auditing", "Behind", "On Track", or "Out". Here, "Behind" refers to learning actions performed belatedly. Prototypical learner behaviours were then identified by clustering the behaviour trajectories (with disengaging behaviour being one of them) [136]. Similar to the approach proposed by Kizilec et al., the skipping predictor relies on sequences of categorized homework submissions.

The predictor was devised and preliminarily evaluated by the author of this thesis while Steven Dostert provided a thorough evaluation in his master's thesis [70], as well as integrating the predictor into Backstage 2 / Projects. The predictor's description and a brief evaluation can be found in [103]. Predictions of skipping together with predictions of examination performance (described in Section 5.2) are used to realise analytics-based nudging (described in Section 4.3). An evaluation on the impact of this learning format on student behaviour can be found in Section 6.2.

Predictor definition. The predictor relies on a Hidden Markov Model (HMM) [199]. HMMs consist of a set of "hidden" (i.e. not directly observable) states, transition probabilities between these states, a set of (directly observable) emissions or signals, and for each hidden state, a probability distribution modelling the probability of observing each emission in this state. After emitting a signal, the HMM changes its state based on the state transition probabilities. Given an HMM and a sequence of signals, the likelihood of the HMM to produce this sequence of signals can be determined using the so-called Forward-Backward algorithm [199].

The HMM used for skipping prediction consists of two states, and the signals **NE**, **OE**, **IK**, and **SKIP**. Training an HMM consists of using the Baum-Welch [16] algorithm to estimate the transition and emission probabilities that would best describe a set of signal sequences (in this case sequences of categorized homework submissions).

Predicting skipping for a student is then performed by analysing their sequence of submitted assignments $s_1, ..., s_n$. Given that these assignments were assessed with categories $c = c_1, ..., c_n$ ($c_i \in \{ne, oe, ik, skip\}$ for $i \in 1...n$), consider the four extended category sequences

$$c_{ne} = c_1, ...c_n, ne$$

$$c_{oe} = c_1, ...c_n, oe$$

$$c_{ik} = c_1, ...c_n, ik$$

$$c_{skip} = c_1, ...c_n, skip.$$

If c_{skip} is the most likely sequence to be produced by the trained HMM, skipping is predicted for the student.

Evaluation. The predictor was trained with data gathered by categorizing homework submissions of 80 randomly chosen students of a course on theoretical computer science [3].

Figure 5.2 shows the trained model. Here, the two states are referred to as *inert* in which **SKIP** is the most probable emission, and *busy* in which **OE** is the most probable emission. Note that both **NE** and **OE** are much more likely to be observed in the *busy* state than in the *inert* state (which motivates the naming). This fits the intuition of "losing momentum" referred to earlier: The hidden *busy* state (or, more precisely, the probability of being in the *busy* state) models the "momentum" a learner currently has while having insufficient knowledge or skipping reduces this momentum.

very poorly performing students, and a "gap" in between.

[3] TCS-2015 see Apendix A.

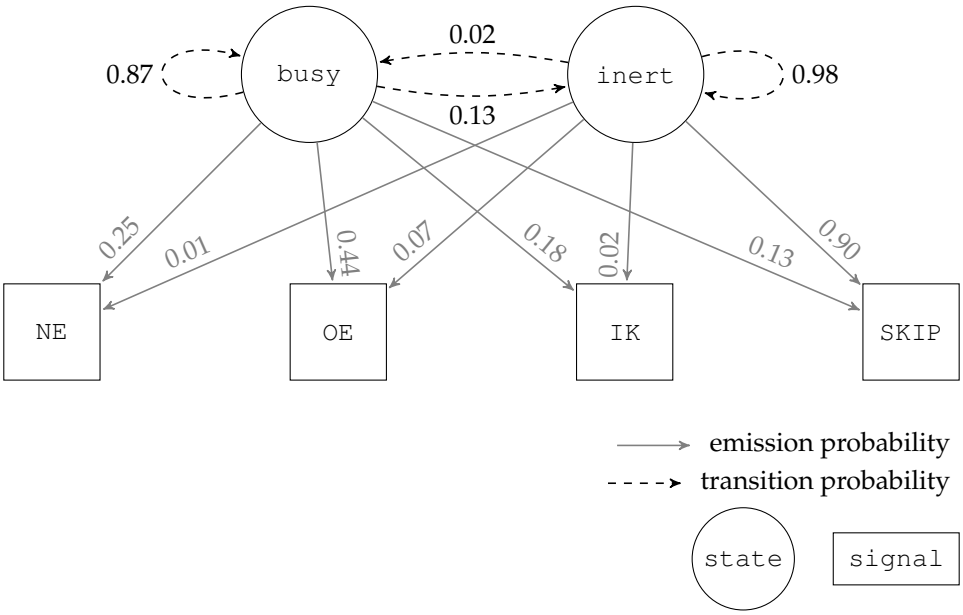


Figure 5.2: The trained HMM, with transition and emission probabilities

An evaluation was performed using 10-fold cross-validation, which yielded an accuracy of 79%, a sensitivity of 73% and a specificity of 85%.

Note that assessment categories were chosen for skipping prediction because they identify lacks of knowledge, which could lead to skipping. To test this hypothesis, a simplified HMM was trained which only knew two signals: "submitted" (observed instead of **IK**, **OE** or **NE**) and "skipped". This model performed slightly a worse in predicting skipping, yielding an accuracy of 0.77%, a sensitivity of 70%, and a specificity of 84%. This confirms the assumption that lagging behind may cause skipping, and that the detection of such behaviour (through **IK** assessments) can lead to a better skipping prediction. Yet, the fact that the simplified predictor only performed *slightly* worse indicates that there probably are more influential reasons for skipping.

Perspective. In the dataset, *skipping* might be a bit overrepresented: If a student decides to drop out of the course (which might be a reasonable choice, if, for instance, too many courses were joined that semester) they are considered to be skipping, and it can be argued whether that is true: The aim of predicting skipping is to deliver interventions which should only be provided to students that are still *possibly* willing to participate. Hence, in the future, the assessment of skipping should be reconsidered, which would probably lead to different predictor properties.

The skipping predictor is independent of the *exact* number of exercises of the course it is used in, yet, it will most likely work best in courses which have a *similar* number of exercises as the course the training data was gathered in. As can be seen by the state transition properties depicted in Figure 5.2, transitioning from *busy* to *inert* is much more probable than transitioning from *inert* to *busy*: The predictor will predict higher and higher probabilities of skipping, even for learners who solve all exercises flawlessly. Note that this rate of "attraction" is specific to the training data, and might change for instance in longer courses. However, that the predictor is robust against small changes in the number of exercises is a desired property, as this number may change between different venues. In the future it might be interesting to compare these models between courses: Do some courses

yield higher "re-entry probabilities" (transitions form *inert* to *busy*) than others?

The predictor is *task-agnostic* as it does not distinguish between tasks. While this allows to change a course's schedule, or even to use the training data on a completely different course, maybe a *task-specific* skipping prediction would be more accurate. Figure 5.1 shows large differences in assessments between weeks: Submissions in week three, for instance, show the largest proportion of *insufficient knowledge* assessments. Accordingly, the increase in skipping in week 4 is one of the largest in the dataset. If such "hurdles" were known beforehand, specific interventions could be implemented. In the evaluated course on theoretical computer science, the homework of week 3 was the first to require the students to independently construct a proof, which was arguable the cause of the rise in *insufficient knowledge* assessments. A *task-specific* predictor would need to hold a representation which *topic* holds particular challenges for the learners.

5.2 Predicting Examination Performance

This section describes a predictor of examination performance which has been briefly introduced in [103].

Predicting examination performances is one of the most common applications of Learning Analytics found in the literature. To make such predictions, a plethora of data sources have been considered, among others: Emotional affects [184], grades in previous courses and demographic data [98, 52], engagement measures [1, 52, 78], homework performance [186, 40], online quiz performances [238], and the use of supplementary learning material [164].

The set of statistical measures employed for examination performance prediction is also quite diverse, ranging from Neural Networks [175] to Decision Trees [98] and Regression Analysis [1, 186, 40].

The predictor introduced in this section uses counts of labels assessing student performances to make predictions. The labelling scheme, which is described in Section 4.3 and is also used for skipping prediction (Section 5.1) encompasses the labels **NE** for "no error", **IK** for "insufficient knowledge", **OE** for "other error", and **SKIP** for skipped activities. As described in Section 4.3 such label counts can be gathered by teachers labelling homework assignments as part of their teaching routine. Multiple choice quizzes, where each possible answer is associated with one of the labels **NE** (correct answer), **IK** (answer indicating a lack of knowledge by the learner) and **OE** (for all other wrong answers), are a second data source. Note that using quiz responses in this manner does only require human labour when designing the quizzes, while homework labelling has to be performed throughout a course.

The examination performance predictor relies on a linear regression model. Linear regression models express a numerical target variable (also referred to as the dependent variable) as a linear combination of different independent variables. For the predictor described in the following, the dependent variable is examination performance (encoded by a mark between 0% and 100%), and the counts of different labels are the independent variables.

Using data gathered from student homework to feed linear regression models to predict examination performances has been attempted before in the literature: Parker et al. used sums of numerical homework assessments as independent variables [186], while Carter et al. used the time spent in different "working states" a student could be in while delivering programming homework [40].

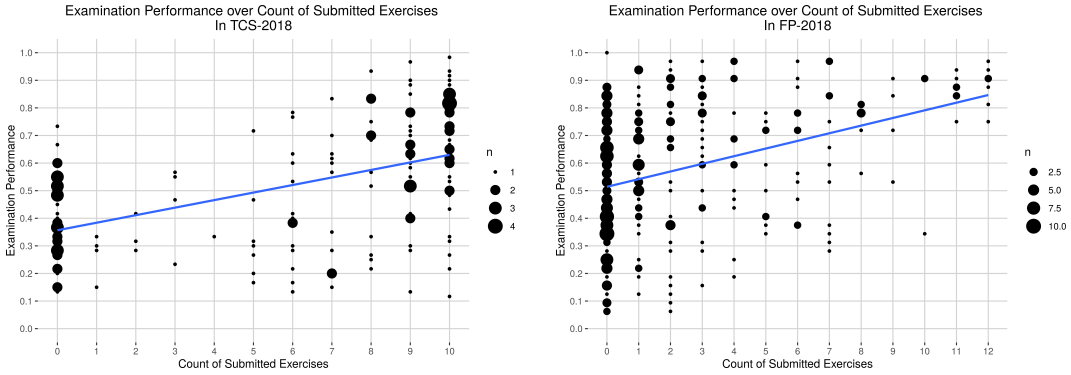


Figure 5.3: Comparison of the influence of homework completion on examination performance. Left: A course on theoretical computer science in 2018 (TCS-2018). Right: A course on Functional Programming in 2017 (FP-2017). Dots represent sets of students, sizes represent set sizes.

Arguably, the relation of homework completion and homework quality on examination success depends on characteristics of the course (the topic, the teaching method, etc.), and obviously on the examination itself. See Figure 5.3 as an illustration of the influence of homework completion on examination success in a course on functional programming[4] and a course on theoretical computer science.[5] In both cases, students delivering more homework performed better in the final examination than their inactive peers: In FP-2017 the average performance of students who skipped at most two assignments is at 1.47 standard deviations above the average performance, these students outperform in average 81% of their peers. In TCS-2017 the figures are similar, strongly participating students (skipping at most two assignments) outperform in average 72% of the other students, their performance is at 1.23 standard deviations above average. In both cases, the linear model (plotted in the figures) predicts a gain of 2.7% per submitted exercise. For both courses, this influence is significant ($p < 0.01$).

Yet, both datasets show remarkable differences: Firstly, homework completion was distributed differently. In FP-2017, 40% of students chose not to do any homework, in TCS-2017 these were only 24%. Secondly, in TCS-2017 there were a considerable number of students *not performing well* in the examination while *having delivered most homework*, in FP-2017 this number is much smaller (visible in the bottom right corners of the plots). And thirdly, average grades in FP-2017 are about 15% higher than in TCS-2018. Considering these facts, linear regression models seem to be suited for examination performance prediction as they perform comparably well on these fairly different datasets.

Also, linear regression models yield *actionable* predictions, as delivering homework always changes the prediction by the same amount. This allows the learner (and the software) to easily predict what *would* happen if the next homework was delivered or skipped.

The predictors of examination performance and skipping can be used together to inform students with a high estimated risk of skipping of the benefit of doing homework. A course format using this technique is described in Section 4.3 and an evaluation of this course format can found in Section 6.2.

Predictor definition. Linear regression models describe a predicted variable Y as a linear combination of several independent variables $X_1 ... X_k$:

$$Y = \beta_0 + \beta_1 X_1 ... \beta_n X_n$$

[4] FP-2017 see Apendix A.
[5] TCS-2015 see Apendix A.

Fitting a linear regression model consists of finding coefficients the $\beta_1 ... \beta_k$, so that the error terms ϵ in the following equations are small for all observed values y of Y and x_k of X_k:

$$y = \beta_0 \cdot x_1 + ... \beta_p \cdot x_p + \epsilon.$$

Accordingly, the examination performance predictor is trained with a set of observed examination results along with counts of assessed labels, yielding a linear model:

$$E = \beta_0 + \beta_1 \cdot X_{ne} + \beta_2 \cdot X_{ik} + \beta_3 \cdot X_{oe} + \beta_4 \cdot X_{skip}$$

Where E is the estimated examination performance and X_l are the counts of observed labels l. Note that β_0, referred to as the intercept, signifies the predicted examination performance if no label counts are available, which is the case at the beginning of a course.

A prediction can be made for a student by counting the labels the student received and calculating E given these counts.

Note that this model violates two requirements often mentioned in the literature: Firstly, the error terms ϵ are usually required to be normally distributed [117]. This requirement cannot hold, because examination marks are bounded (by 0 and 1). For very large or very small predictions (near 0, or 1) the error ϵ can only assume small values because values above 1 or below 0 can not be observed. Hence, ϵ cannot follow a normal distribution which would allow any value. Secondly, the independent variables are required to be *statistically independent*. This is not the case here as labels are disjoint classifications of student behaviour and exclude each other.

Evaluation. The quality of the examination performance predictor was evaluated with two datasets. The first dataset encompasses data from 11 labelled homework assignments of 82 randomly chosen students in a course on theoretical computer science[6]. The homework assignments were tasked every week over the period of the whole course. This dataset is referred to as the "homework dataset". The second evaluation used data gathered from online quizzes using the audience response system of Backstage 2 in a course on functional programming[7]. This dataset encompasses data of 253 students. Quizzes were conducted on a weekly basis throughout the course and served to recapitulate the content of the last lecture. In total, 10 quizzes were conducted (with two of the course's weeks encompassing no quizzes), yet due to the internet connection in the auditorium failing in three occasions, only data from 7 quizzes could be recorded. This dataset is referred to as the "quiz dataset".

With the homework dataset, **SKIP** (missed learning activity) and **IK** (insufficient knowledge) impact on examination fitness with a significance level of 0.05, while the inclusion of the other variables does not worsen the predictor's accuracy. The final model E_{hw} trained on the homework dataset yields the following coefficients:

$$E_{hw} = 0.57 + 0.019 \cdot X_{ne} - 0.018 \cdot X_{ik} + 0.007 \cdot X_{oe} - 0.006 \cdot X_{skip}$$

The R^2 error of E_{hw} is 20%. Note that a linear model only accounting for homework completion (similar to the model seen in Figure 5.3) performs worse, with an R^2 error of 14%.

With the quiz dataset, **OE** (other error) and **IK** (insufficient knowledge) impact on examination fitness with a significance level of 0.05. This dataset contained no **SKIP** labels, because of software failures leading to gaps in the data (whether students skipped in these gaps cannot be determined). Nevertheless, the model model trained on the quiz dataset E_{quiz} has an R^2 error of 19.8%, which comparable to E_{hw}.

$$E_{quiz} = 0.67 + 0.02 \cdot X_{ne} - 0.008 \cdot X_{ik} + 0.003 \cdot X_{oe}$$

[6] TCS-2015 see Apendix A.
[7] FP-2017 see Apendix A.

As with the homework dataset, using label counts as independent variables yields a better predictor than using quiz correctness alone. The linear model using only quiz correctness only exhibited an R^2 error of 10%.

Perspective. Choosing the set of independent variables used for prediction can not only change the predictive qualities of the predictor but also its pedagogical use. If only the count of submitted assignments is used, β_0 (the estimation displayed at the beginning of the course) estimates the examination performance when *no* homework is delivered, with predictions improving by submitting homework and remaining stable by skipping. If, in contrast, only the count of *skipped* assignments was used, β_0 signifies the estimated examination performance if *all* homework was delivered, with predictions *decreasing* by skipping and remaining stable by delivering homework. Both models describe the data equally well, as the independent variables are inversions of one another, but both models might be chosen with different pedagogic intentions: The first model might be perceived as an encouragement to do homework and as a reification of the work done so far. The second model might be perceived as a punishment for skipping homework while exploiting the endowment effect.[8]

The predicted values computed with E_{hw} can both increase and decrease depending on a student's behaviour: Skipping and exhibiting insufficient knowledge decrease predicted performances while making no errors or other errors increases the predicted values. Therefore, predictions made by this model can both be perceived as encouragement and punishment. Yet, by inverting the respective variables, the model could be easily adapted to suit only one of these "pedagogical modes". An interesting approach would be to let students choose which model they want to receive predictions from: Some students might prefer to have an optimistic model, starting with high predictions which can decrease along the course, while others might prefer to see their estimated score increase as they deliver homework.

The evaluation showed that models using label counts for prediction outperform models which simply use counts of delivered homework or quiz correctness. Future developments could aim to further increase the predictor's accuracy. One approach could be to find shared behaviours of students who delivered homework or made quizzes while receiving grades that were lower than the predictor estimated. These students could not benefit from delivering homework or making quizzes as the others, and integrating data on these behaviours into the model could both increase the predictor accuracy and, hopefully, the learning efficiency.

The current predictor only accounts for homework quality but not for the feedback the students received *on their homework*, which might overemphasize the severity of making errors. Indeed, feedback quality can make the difference between changing misconceptions and maintaining them until the examination. Obviously "receiving good quality feedback" is not in the power of the student, but in Backstage 2 / Projects students can ask questions on their feedback and finally indicate their understanding. If the teaching staff can assure that student questions on feedback are answered until the student is satisfied with the answers, data gathered in this way could also be integrated into examination performance predictions. This would emphasize that both the teacher *giving* feedback, and the students making an effort in *understanding* feedback are part of the learning process.

5.3 Predicting Systematic Errors and Misconceptions

The predictor presented in this section aims to predict systematic errors and misconceptions a student will make in a course, which is, to the author's best knowledge, a novel application of Learning Analytics. As the predictors presented in the previous sections, it relies on

[8] The preference of people of keeping possession over acquiring new possession [125].

sequences of labelled homework submissions, where labels are used to model systematic errors and misconceptions.

The misconception predictor was conceptualized and preliminarily evaluated by the author of this report, but Andreas Born integrated it into Backstage 2 / Projects and evaluated and optimized its performance. A thorough description of the implementation and the optimization process can be found his thesis [26], a brief description of the predictor and its properties can be found in [103].

The predictor relies on collaborative filtering, a technique often used for product recommendation e-commerce systems. In such systems, users rate products either explicitly by awarding points or stars, or implicitly (e.g. by buying a product or watching a film). Collaborative filtering identifies similarities between users by identifying similarities in the ratings they gave (whether they liked or bought the same products). The intuition is that if two users bought or liked the same products in the past, they will probably like and buy the same products in the future: If one them buys a specific product, it can be recommended to the other. Similarly, one could assume that if two students made the same *errors* in the past, they might make the same errors in the future. This intuition is underpinned by the notion of latent variables, which are common personal properties which are not directly observable but can be used to explain a large variety of behaviours [27]. In e-commerce, a latent variable could be the preference for certain book genre or a specific hobby which leads a group of people to prefer certain products. In the case of systematic errors in education, a latent variable could be a shared fundamental misconception or missing preliminary knowledge. The basic data structure used for collaborative filtering is the so-called rating matrix $R \in M_{n,m}$, which contains for each of the n users and each of the m products the rating the user gave to the product, or no value if the user did not rate that product. The "missing" ratings are typical for explicit rating data (as users usually cannot have opinions on all products available), and collaborative filtering approaches often try to predict ratings a user *would give* to a product if they bought it. Recommendations are then compiled from products the user did not (yet) buy that have the highest *predicted* rating.

The misconception predictor uses a label matrix (similar to a rating matrix) containing for each student and each known error whether the student made the error (denoted by a 1), did not make the error (denoted by a 0), or no data if the value cannot be specified. A specific systematic error can only be observed if a regarding exercise was assigned and submitted by a student. Therefore, missing values in the label matrix occur if students skipped assignments, or if a regarding exercise was not yet submitted.

A technique related to collaborative filtering is frequent pattern mining, where sets of items that are usually bought together are recorded. Figure 5.4 depicts a section of the label matrix generated from homework submitted by students in a course on theoretical computer science in 2015, with a notable frequent pattern.

Predictor definition. A common practice for collaborative filtering is to find and approximate a decomposition of the rating matrix $R \in M_{n,m}$ into two smaller matrices $U \in M_{n,k}$ and $P \in M_{k,m}$, where $R \approx U \cdot P$ [266].

The rank $k \in \mathbb{N}$ of U is referred to as the number of latent or hidden variables, and U and P as user and product features respectively. These notions are sensible because, if $U \cdot P$ estimates R well, then the rating behaviour of user i (encoded by the row i of R, $r_{i,\star}$) can be approximated well by a linear combination of rows of P:

$$r_{i,\star} \approx u_{i,\star} \cdot P = \sum_{j=1}^{k} u_{i,j} \cdot p_{j,\star}$$

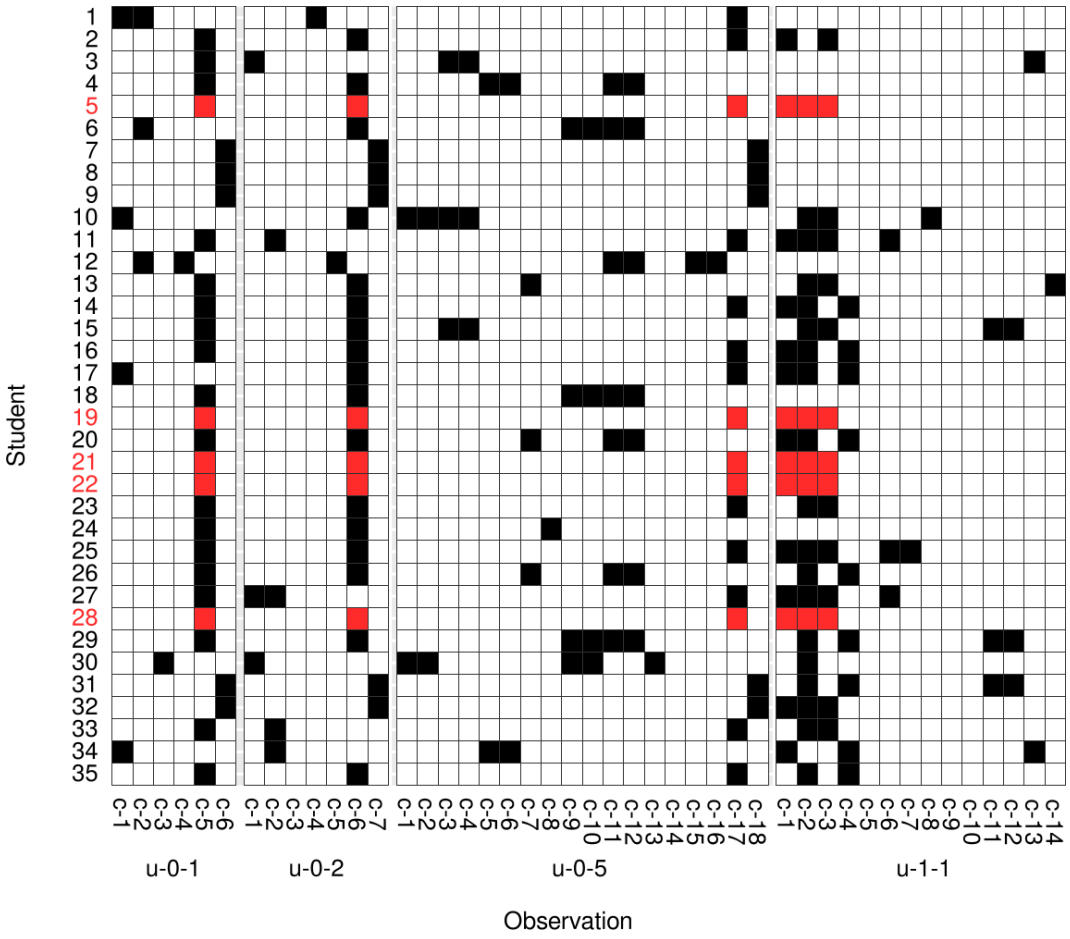


Figure 5.4: A section of the label matrix generated from homework submissions in a course on theoretical computer science (TCS-2015). The rows represent students, the columns represent systematic errors. Systematic errors are grouped by the exercise they can occur in (u-0-1, u-0-2, u-0-5, u-1-1). A matrix entry is white if the error was not made by a student, and coloured if it was made. The coloured entries represent a frequent pattern shared by five students.

In this sense, the rows of P represent *prototypes* of rating behaviour, and U estimates to which degree each user *fits* each prototype.

As stated above, R may contain missing values, so finding an approximate decomposition of R refers to finding two matrices U and P whose product approximates the *known* values of R. To find such a decomposition, Zhou et al. propose the Alternating-Least-Squares with Weighted-λ-Regularization (ALS-WR) algorithm [266], which tries to minimize the following cost function f

$$f(U,P) = \sum_{(i,j)\in I} (r_{i,j} - u_{i,\star} \cdot p_{\star,j})^2 + \lambda \left(\sum_{i=1}^{n} \overline{r_{i,\star}} \, ||u_{i,\star}||^2 + \sum_{j=1}^{m} \overline{r_{\star,j}} \, ||p_{\star,j}||^2 \right)$$

Where I is the set of indices of known ratings in R, $\overline{r_{i,\star}}$ is the number of ratings given by user i, $\overline{r_{\star,j}}$ is the number of ratings product j received, and λ is a numerical regularization parameter. The second term of the function, which is weighted by λ, penalizes large parameters in U and P, and is used to prevent overfitting.

Numerical solutions for U and P are then found by first filling both matrices with random values, and then using the gradient descent technique to alternatingly improve the estimation of U while fixing entries of P and then improving the estimation of P while fixing entries in U. Note that the algorithm ignores the "missing values" in R (they do not occur in the cost function) while producing matrices P and U *without missing values*. The entries of $P \cdot U$ can then be used as predictions of ratings "missing" in R.

Note that the number of latent variables k and the regularization parameter λ have to be estimated for this algorithm.

To predict systematic errors a student s would make when attempting a specific exercise, a personalised label matrix L_s is constructed for s by adding a row encoding the systematic errors s made so far to a (previously collected) label matrix L. Then, predictions of error occurrences are made for s by using the ALS-WR algorithm described above and using L_s as the rating matrix. From all systematic errors which can occur for the exercise in question, the errors with values exceeding a chosen threshold t are chosen as prediction.

Choosing a set of predicted errors with an occurrence probability above t is used false positives: Possibly, a student will not make any systematic error (in contrast, product recommendation sites typically assume that all users will buy products).

Evaluation. For the evaluation of the predictor, homework submissions of 82 randomly chosen students of a course on theoretical computer science [9] were labelled. This dataset is described in Section 4.2 where the labelling validity is reported.

For each exercise, 10-fold cross-validation was performed: The data of 90% of the students were chosen as training data, yielding a label matrix L. For each of the remaining students, L_s was constructed by adding all known labels of the student *up to* the exercise in question while marking the rest of the labels as unknown. The prediction was then performed as described above, and compared to the observed labels.

This was repeated for different configurations of parameters k (the number of latent variables) and λ (the regulation parameter), yielding an optimal model with parameters $k = 40$ and $\lambda = 0.065$. Figure 5.5 shows the performance of this optimized model with varying thresholds t in a ROC curve. The results are promising: The ROC curve diverts notably from the diagonal (which would signify random guessing), and an optimal trade-off can be found with a threshold 0.05 yielding a specificity of 80.7% and a sensitivity of 71.8%.

[9] TCS-2015 see Apendix A.

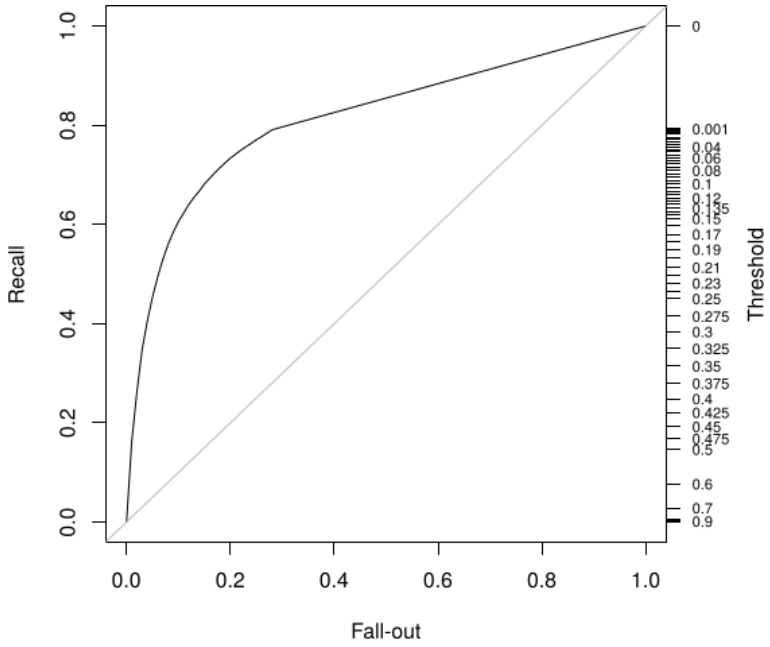


Figure 5.5: ROC curve of the optimized misconception predictor for various thresholds, taken from [26].

It is noteworthy that more frequent misconceptions are predicted with higher accuracy than less frequent misconceptions, which can be seen in Figure 5.6. Possibly, the predictor would perform better with training data of more students (yielding higher misconception frequencies).

Perspective. The misconception predictor is interesting because it delivers data directly providing "feed-forward" (i.e. an indication "what to do next"), while other learning analytics (such as the predictors described above) rather present assessments such as predictions of examination performance, based on the learning behaviour shown so far. In the future, such predictions could be used for learning personalization, for instance by prompting students with descriptions of systematic errors and possibly additional material chosen specifically for a student when they attempt an exercise.

Yet, as shown in Section 6.1, simply by providing all students with descriptions of *all* known systematic errors without personalization, the occurrence of these errors can be drastically reduced. Indeed, introducing a predictor bears the risk of making false predictions: Prompting students with error descriptions which are not helpful for them, or (which is arguably worse) failing to prompt error descriptions which would have been helpful. If there are few systematic errors known for each exercise, prompting all error descriptions to all students seems to be the "safer" option.

A different use of such predictions would be to identify *causes* of systematic errors. If certain errors often occur in combination, they possibly share an underlying cause. Such an analysis could be achieved by presenting teachers with sets of correlating errors (possibly derived from the factor matrix P described above) and asking the teachers to identify similarities between these errors. Identifying underlying causes of errors would be a promising approach to improve teaching.

In the future, the implementation of the predictor could be improved. Firstly, its current implementation is not very efficient: Matrix factorizations are computed for each student

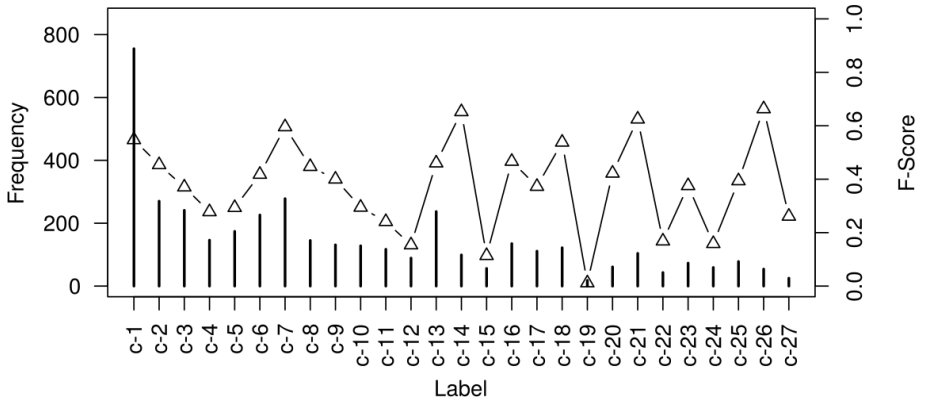


Figure 5.6: Comparison of the F-scores (triangles) and frequencies (bars) for the 27 most frequent misconceptions, showing a notable correlation. Image taken from [26].

and exercise, while they could be stored and re-used. Secondly, the predictor currently models exercises and errors as unique entities which share no properties other than their co-occurrence: If a new exercise is introduced to the course, the predictor cannot predict systematic errors for it, even if certain predictions would be sensible: Many systematic errors of, for instance, proofs by induction may occur in different exercises requiring this technique. Such an extension would require some kind of ontology for exercises, topics, and associated systematic errors.

5.4 Predicting Levels of Programming Competence

The predictor presented in this section aims to predict the programming competence of students. To assess programming competence, source code is categorized into three levels:

1. Code written in a correct file format.
2. Syntactically correct code.
3. Semantically correct code.

The predictor estimates for each competence level the likelihood of a student being able to produce code of that level. As an input, the predictor uses sequences of categorized code submissions.

This level scheme is motivated by observations reported in Section 4.4: In a peer-taught course on functional programming (FP-2017-p), submitted programming assignments were automatically classified in this manner. For all competence levels, examples of code *reaching the level*, and examples of code *failing the level* were observed.

Competence levels are inclusive: Syntactically correct code is assumed to be provided in a correct file format, and semantically correct code is assumed to be syntactically correct. Note that with this assumption, a student's competence may be underestimated: Source code embedded in, for instance, a PDF file may be semantically correct – and the student was able to write such code. Also, if a student writes a semantically correct program and accidentally deletes one character from the document, resulting in a syntactically incorrect program, it can be argued whether that student *really* does struggle with the syntax barrier. Yet, enforcing a strict hierarchy eases the automatic categorization of source code using the language's compiler or interpreter.

Similar categorization approaches can be found in the literature: Carter categorized code written by students while working on assignments by defining four "states" resulting from combinations of "syntactically correct" / "syntactically incorrect" and "semantically unknown" / "semantically incorrect" [40]. "Semantically unknown" refers to code which runs without throwing any runtime errors, which is interpreted by the authors as an indication for semantic correctness. One of the states proposed by Carter is "semantically unknown and syntactically incorrect". In Carter's model, this state is reached by introducing a syntax error to a program which was previously semantically correct. Hence, to describe the process of programming, this state may be sensible. Yet, to describe a submitted program (the *outcome* of the programming process), such a category is less sensible because a program that cannot be interpreted cannot "work as intended".

Another code categorization approach was performed by Denny et al. who categorized code as "passing all tests" (similar to level 3), "compiling but not passing all tests" (similar to level 2), and "not compiling" (similar to level 1) [63]. Note that this categorization scheme does not assess code submissions "in the wrong file format" as the software used in [63] would not allow such submissions. Similar to the evaluation presented in Section 4.4, Denny et al. found a large number of students struggling with the syntax of the taught programming language. Yet, different to the results presented above, Denny found that "compiling code failing the provided tests" (level 2 but not 3) was more common than code "passing all tests" (level 3). Interestingly, the evaluation presented in Section 4.4 yielded the opposite result: Most students who were able to write syntactically correct code, also wrote semantically correct code. This difference might be caused by the different programming languages taught: Haskell (taught in the course described in Section 4.4) being declarative and functional, and Java (taught in the course evaluated by Denny et al.) being imperative and object-oriented. This can be explained intuitively: Programs in a functional and declarative language are correct if the programmer managed to correctly *declare* what a solution to a problem is (which encompasses syntax and semantics *simultaneously*), while a program in an imperative programming language is correct if the programmer formulated syntactically correct instructions for the interpreter and these instructions "realise the right thing" upon interpretation.

The predictor presented in this section can model such "hurdle situations" where mastering one level often leads to the mastering of the next level.

The competence level predictor was devised and preliminarily evaluated by the author of this thesis, while Robert Pospisil provided the implementation as well as the idea of making *task-specific* predictions (described below). A more thorough evaluation and the documentation of the implementation can be found in his master's thesis in [193].

Predictor definition. The competence level predictor relies on a Bayesian network, which consists of a set of random variables, and a set of statistical dependencies between these variables. If some of the variables (referred to as the evidence variables) are observed, inferences can be drawn about the probability distributions of the other (unobserved) variables.

Bayesian networks are usually visualized by directed acyclic graphs, where a node represents a variable, and an edge represents statistical dependence between two variables (an edge from variable a to variable b is read as "b depends on a"). Every variable is associated with a random distribution, where variables depending on other variables are associated with conditional distributions. Because all variables of the Bayesian network described here can only assume a finite set of values, random distributions can be displayed by probability tables (in the case of independent variables) or conditional probability tables (in the case of dependent variables).

Figure 5.7 depicts the whole Bayesian network used for prediction. It consists of three types of variables:

- The three unobserved **competence level variables** "form", "syntax", and "semantic" model the probability of a student submitting code satisfying or failing the respective level. These variables can assume the values "correct" (reaching the level) or "incorrect" (failing to reach the level).
- The three **history variables** represent assessments of previous student submissions, and can assume the values "positive", "neutral" or "negative". For instance, observing a "negative" syntax history for a student signifies that the student struggled often with the syntax of a programming language in the past. Observations of history variables are made by analysing all previous code submissions of a student. Consider a sequence $s_0 \dots s_n$ of student code submissions where s_0 is the first (hence oldest) and s_n is the last (hence newest) submission. From these, submission level scores $l_0 \dots l_n$ are computed by mapping each submission satisfying a given level to 1 or -1 respectively. The level score L is then defined as a weighted sum of submission level scores:
$$L = \sum_{i=0}^{n} 2^{i-n} l_i = \frac{1}{2^n} l_0 + \ldots + \frac{1}{2} l_{n-1} + l_n$$
Note that the weighting emphasises newer submissions over older submissions. If L is smaller than -0.5 a "negative" level history is observed, if it is larger than 0.5 a "positive" level history is observed, and a "neutral" level history is observed otherwise.
- The **difficulty variable** models the difficulty of the courses exercises. The difficulty of an exercise is defined by the number of students being able to correctly solve the exercise (submitting a semantically correct solution). If 66% or more of all students can solve an exercise, it is regarded as "easy"; if only 33% of students or less can solve an exercise, it is regarded as "difficult"; and it is regarded to be of "medium" difficulty otherwise.

The following dependencies are introduced between the variables:

- Each competence level variable ("form", "syntax", and "semantic") depends on its respective history variable (following the intuition that recently having struggled with a specific problem influences the current performance).
- The competence level variables "syntax" and "semantic" depend on the variables modelling the next lower level. This allows to model hurdle situations as mentioned above.
- All competence level variables depend on the difficulty variable.

The model is trained by using student code submissions of a course, counting the occurrences of the competence levels for each student, and computing the (conditional) probability tables which completes the Bayesian network.

Predictions for a student are made by computing the history assessments as described above for each level, entering the computed assessments as observations in the trained Bayesian network, and inferring the competence level variables (for instance using variable elimination [265]).

Additionally, if the difficulty of an attempted exercise is known, it can be also entered as an observation of the "difficulty" variable. Note that this additional step performs a *task-specific* prediction: Estimating the likelihood for a student to reach a certain level *given a specific exercise*.

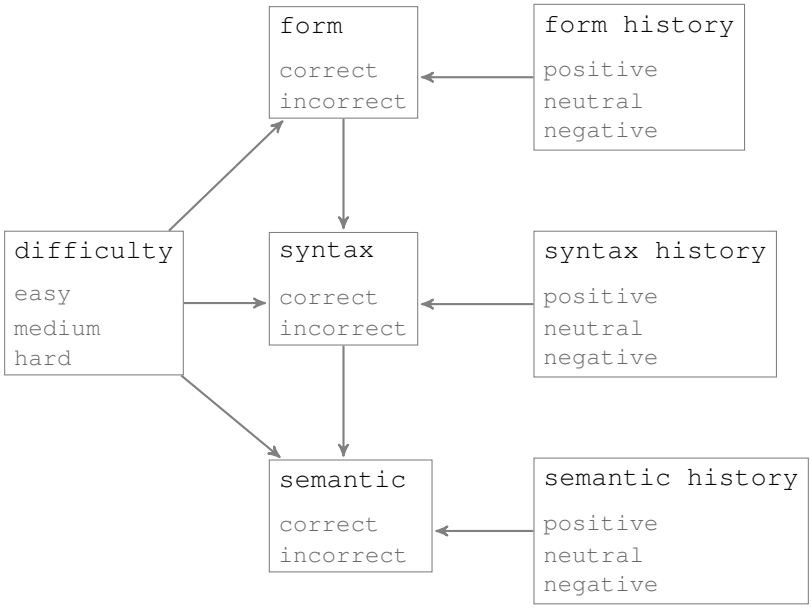


Figure 5.7: Complete Bayesian network of the competence level predictor. Possible values a variable can assume are displayed in grey below the variable names.

	task-specific	task-agnostic
accuracy	0.82	0.71
sensitivity	0.78	0.64
specificity	0.69	0.75
F1-score	0.74	0.43

Table 5.1: Performance of the competence level predictor. The *task-specific* predictor uses difficulty observations of the attempted exercise, while the *task-agnostic* predictor does not.

Finally, it is predicted that a student reaches a competence level in their next assignment if the probability of the regarding competence level variable to assume the value "correct" exceeds a chosen threshold.

Evaluation. The predictor was evaluated with data gathered in a peer taught course on functional programming (FP-2017-p, introduced in Section 4.4). For the evaluation, 10-fold cross-validation was used.

Table 5.1 reports the average accuracy, sensitivity, specificity, and F1-score the predictor exhibited both for the *task-specific* usage (using observed exercise difficulties for predicting) and the *task-agnostic* usage (leaving the difficulty variable unobserved). The predictor benefits from the use of difficulty observations, improving the accuracy by 11%. Interestingly, the *task-specific* predictor is more sensitive than specific, while the *task-agnostic* predictor is more specific than sensitive: Knowing the difficulty of an exercise aids at identifying the competence levels a student will reach, while hindering the correct identification of levels a student will fail.

Perspectives. In the current form, the predictor showed quite promising properties, with an accuracy of 82% in the *task-specific* use and 71% in the *task-agnostic* use. Yet, certain parameters such as the weighting coefficients of the level history assessment were arbitrarily chosen. Future developments could try to improve these parameters.

Note that the nature of a Bayesian network allows making both *task-specific* and *task-agnostic* predictions in the same course without changing the implementation: Difficulty observations of exercises could be used if available, with the predictor still being functional if they are not. In this way, new exercises can be added to a course, with the predictor taking advantage of the assessed difficulty of old exercises while still delivering predictions for the new exercises. This is a clear advantage of the competence level predictor over similar approaches such as Rash models (introduced in [161]) which rely on data on all attempted tasks being available before making a prediction.

In the future, competence level predictions could be used to implement personalized scaffolding for programming assignments: If a user is predicted to struggle with the syntax of a programming language, the software could provide them with additional aids, such as worked examples, or even providing a block-based interface (famously implemented by the scratch programming language [158]) instead of a text editor.

Another use of the competence level predictions would be to realise "smart" orchestration of peer teaching: Students predicted to master, for instance, the syntax level of a language could help students which are predicted to struggle with this level. In this way, the students' abilities could be used efficiently, assuming that all students who mastered a competence level could help students who have not yet reached that level.

CHAPTER 6

Fostering Self-Regulation and Exploratory Learning

This chapter presents results on the improvement of learning behaviours and learning outcomes caused or sustained by Backstage 2 / Projects. In Section 6.1 results regarding conceptual change are reported, Section 6.2 discusses observed changes in the students' learning behaviour, and Section 6.3 reports on a laboratory study aimed at sustaining exploratory learning of STEM's formal languages. Each of these sections is concluded with perspectives for software improvements or further research. In Section 6.4, the results are interpreted with regard to self-regulated learning and teaching efficiency.

6.1 Fostering Conceptual Change

The conceptual change model describes learning as a process in which students change their existing (possibly erroneous) beliefs based on the experiences they have. Fostering conceptual change is a well established and effective teaching method: Hattie found conceptual change programs among the most effective teaching methods (average reported effect size Cohen's $d = 0.99$) [101], and Harasim presents conceptual change as a central component of her "Online Collaborative Learning" theory [100]. Section 3.3 gives a brief overview of related work on the topic, as well as discussing different uses of the terms *systematic error* and *misconception*.

This section discusses two software-driven measures aimed to foster conceptual change: Collaboratively-generated refutation texts, and a specialized scaffolding embedded in the platform's code editor Knoala (described in Section 4.1).

Many software-driven interventions aiming to induce conceptual change found in the literature address misconceptions which were established *before* the interventions for a population of students [15, 116, 115]. One contribution of Backstage 2 / Projects is its support for the collaborative *collection* of misconceptions (see Section 4.2). Note that a list of common misconceptions along with their prevalences is a simple yet very powerful form of Learning Analytics. It is (in part) due to this collaboratively generated data that the measures discussed in this section could be implemented and evaluated.

Collaboratively-generated refutation texts. The evaluation presented in Section 4.2 showed that the use of collaboratively collected descriptions of systematic errors was well accepted among students: They perceived these descriptions, which were presented alongside their exercises and were often written as "refutation texts" [240], as helpful for their learning. The data presented in the following shows that this measure was not only *perceived* as helpful, but *did help* students in avoiding systematic errors. This result was published in [104].

Data gathered in three consecutive courses on theoretical computer science (TCS-2016, TCS-2017, TCS-2018[1]) was evaluated to measure the effectiveness of the intervention. The first dataset gathered in TCS-2016 is referred to as the *base-dataset*. It was collected to initially establish which systematic errors occurred in the course as well as to determine their occurrence frequencies (see Section 4.2 for a description of this process).

The second dataset, gathered in TCS-2017 and referred to as the *validation-dataset*, was collected to establish whether systematic errors would re-occur with similar frequencies in the following course venue. The third dataset, referred to as the *primed-students-dataset*, was gathered in TCS-2018. In this course, students were provided with error descriptions while working out homework exercises as described in Section 4.2. This intervention aimed at reducing the occurrence rates of the displayed errors.

Many experiments on conceptual change are conducted in laboratory settings with pre and post-tests to establish whether a teaching method administered between tests was effective in reducing misconceptions. The research presented in this section does not follow this approach. While laboratory studies have the important advantage of providing controlled variables, the design used here can provide insights on the learning of far more students over a longer period.

Error occurrences in the *base-dataset* and the *validation-dataset* correlated positively ($k = 0.49$), which indicates that systematic errors were made with similar frequencies in these courses. The correlation between error occurrences in the *base-dataset* and the *primed-students-dataset* was also positive but smaller ($k = 0.21$). This indicates that systematic error occurrences changed due to the implemented priming in TCS-2018. Yet, the correlation between the *base-dataset* and the *primed-student-dataset* is still significantly positive: Common errors in TCS-2016 were still common in TCS-2018. Possibly, the implemented measure affected only certain students.

In TCS-2016, 35% of all homework submissions contained systematic errors. In TCS-2018, this number dropped to 18%, which is a significant change ($p < 0.01$). To further evaluate *which* students avoided systematic errors, the *primed-student-dataset* was split by student activity into four equally sized sets. Here, student activity was assessed by the number of hours in which an action of a student was registered on the platform. The results of this evaluation are shown in Figure 6.1: Student activity had a large impact on the likelihood of making systematic errors. Only 14% of the homework submitted by students in the "most active quartile" contained systematic errors. In contrast, 31% of the homework submitted by students in the "least active" quartile contained systematic errors. Note that the rate of systematic errors for the least active students in TCS-2018 is comparable to the average rate of systematic errors of *all* students in TCS-2016 (35% compared to 31%).

The fact that the most active students made the fewest systematic errors strongly indicates that the overall drop in systematic errors in TCS-2018 was indeed caused by the implemented intervention, as the least active students in TCS-2018 (who rarely visited the platform) made nearly as many systematic errors as the students TCS-2016 which were instructed without the intervention.

[1] See Appendix A for descriptions of these courses

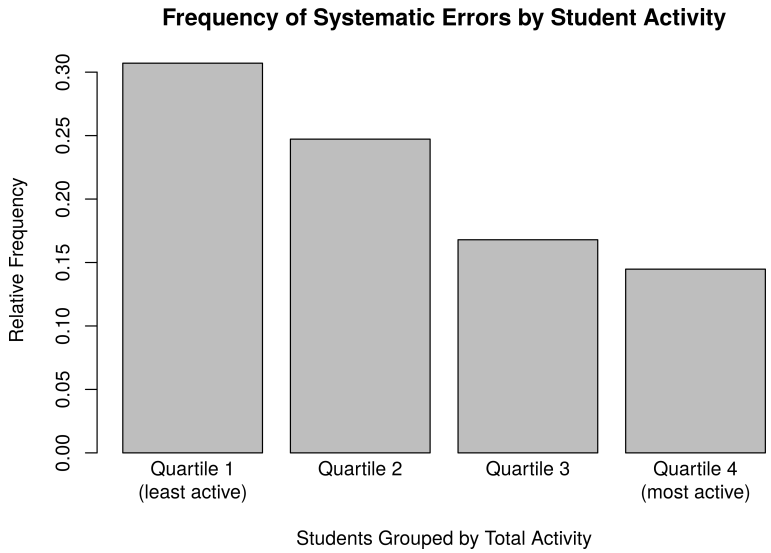


Figure 6.1: Relative frequency of occurrences of systematic errors, by student activity. Quartile 1 encompasses data from the least active quarter of students, quartile 2 data from the second least active students, etc. Image taken from [104]

Scaffolding. The courses on theoretical computer science evaluated for this research required students to learn a set of "theoretical programming languages" (described in Section 4.5), two of which, referred to as LOOP and WHILE, are discussed in greater detail in the following. The imperative programming languages LOOP and WHILE follow mainly the same syntax while providing different means to realise repetition: WHILE allows to specify "while-loops" which are executed until a certain condition is met, and LOOP only allows to specify fixed-length loops [222, Chap. 2.3].

As for all other topics of the course, systematic errors also occurred during the learning of these languages and were collected on the platform. Even though the online code editor Knoala supported students in delivering homework for these topics, syntax errors were very common in the submitted homework.

The most common systematic error observed for these topics was a syntax error caused by an incorrect placement of semicolons. In LOOP and WHILE, two statements are *separated* by a semicolon, while in many of today's practical programming languages (java, JavaScript, Python, C, etc.) semicolons are used to *end a statement*. This means that placing semicolons in LOOP in the same ways as in more "common" programming languages (like the ones the students are used to) causes syntax errors.

It is noteworthy that the special semicolon placement in LOOP and WHILE is somewhat arbitrary (possibly it was chosen to resemble older programming languages like Pascal [118]), and an equivalent definition of these languages with "more common" semicolon placement could be devised easily. Fortunately for this research, such an adaptation was *not* attempted, which allowed observing conceptual change among the students: Starting with a concept like "semicolons are always used to end a statement" and ending with a concept like "In LOOP and WHILE, semicolons are used to separate statements". From an educational point of view, it can be argued whether fostering this particular conceptual change is worth an educator's or a student's time as the difference seems rather benign. Yet, computer science students, and arguably all students studying STEM fields, regularly have to learn new formalisms, many of which may seem unintuitive at a first encounter.

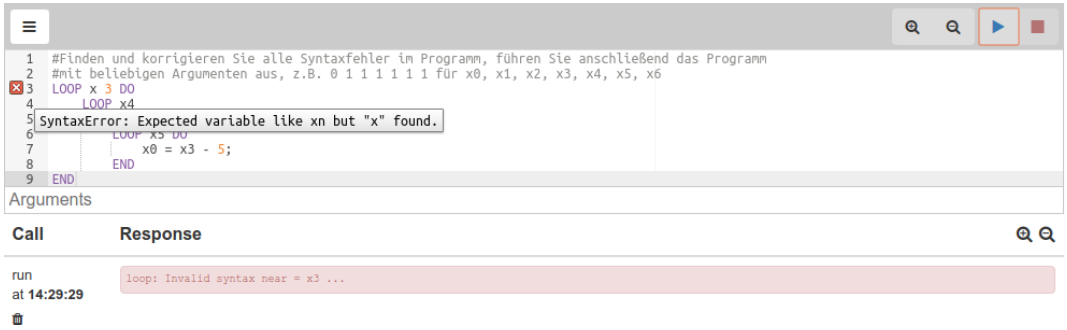


Figure 6.2: Knoala containing an erroneous LOOP program. The implemented scaffolding provides the red error mark left of line 3, and an tooltip text made visible by hovering the error mark.

The syntax errors Knoala provided for these languages were rather limited as they were only displayed after executing a program (while many development environments constantly verify code syntax), and did not indicate the line of text in which an error occurred. For her bachelor thesis, Galina Keil extended Knoala with a scaffold component for LOOP and WHILE, consisting of improved error messages and a continuous code analysis [130], which are shown in Figure 6.2. The following paragraphs report results gathered in a laboratory study which evaluated these improvements.

One goal of the laboratory study was to assess whether students who were provided with the scaffolding would learn the "unintuitive" placement of semicolons in LOOP and WHILE better than students who were not. Other results of this case study are discussed in Section 6.3.

The study followed an AB test design and was conducted with 36 participants of which 16 where assigned the *scaffold* condition and 20 the *default* condition. All students heard a short lesson on the definition and use of the programming languages LOOP and a lesson on the definition and use of the programming language WHILE. After each lesson, students were asked to solve one code correction exercise, in which they were provided with an erroneous program they had to correct, and two programming exercises, in which they had to write a program fulfilling a given specification. All students were asked to use Knoala to work out their solutions, yet only students in the *scaffold* condition were provided with the additional scaffolding.

"Scaffolded" students less often attempted to run code containing syntax errors (including the "semicolon-error" in question) while working out their solutions. This is not surprising, as the implemented scaffold immediately reported syntax errors, while the "unscaffolded" students had to run their programs to be informed about them.

In a post-test, all students were asked to correct a given program which contained, among others, an "incorrectly placed semicolon" as described above. In this post-tests students had no access to Knoala. Students understanding the "unintuitive" use of semicolons in LOOP and WHILE would have no problem finding this error. Indeed, students who learned *with scaffolding* identified this error significantly more often ($p < 0.05$), which is illustrated in Figure 6.3: Arguably, the implemented scaffolding helped the students in learning the unintuitive concept, and hence it fostered conceptual change.

Perspectives. This section discussed two technology-driven interventions which fostered conceptual change, one through displaying refutation texts of systematic errors and misconceptions, and one through a scaffolding embedded in a code editor. Yet, before these interventions could be implemented, the systematic errors the interventions addressed had

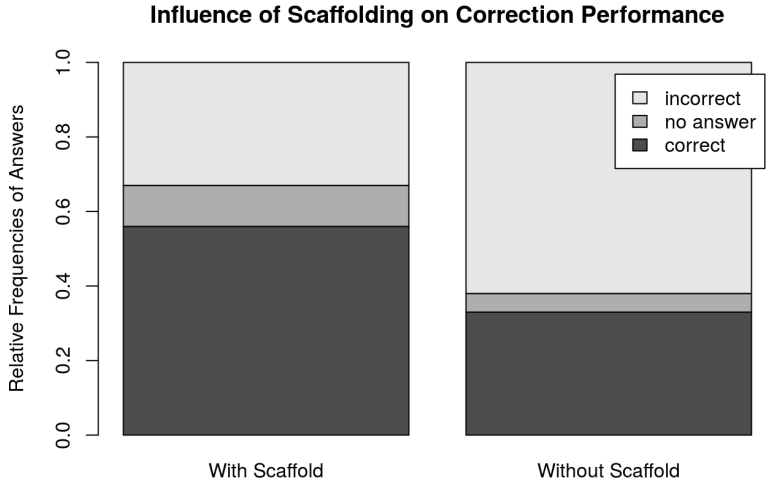


Figure 6.3: Relative frequencies of the results of the code correction task in the post test. Correct answers correctly identified the "semicolon error" and gave a correct explanation for its occurrence.

to be *identified* – a task which, in sum, required at least as much work as implementing the software which delivered the interventions.

A perspective for future work is to identify specific systematic errors and implement technology-driven interventions in other courses as the evaluation presented here solely relies on data gathered in courses on theoretical computer science. Yet, if homework submissions are to be used as a data source, gathering a sufficient amount of data can be a problem. As noted in Section 4.2, a teacher would have to review 52 homework submissions to be 90% sure to identify an error occurring in 10% of the submissions. Besides having to do this substantial amount of work, one has to assert that there are so many submissions to revise! In several examined courses (LDM-2018, LDM-2019, FP-2017[2]), systematic errors could not be sensibly identified because homework delivery rates were too low (even though the courses encompassed several hundred students each). Note that the general problem of skipping homework is discussed in Section 4.3, and attempts in solving this problem are discussed in Section 6.2.

If however such an error analysis *can* be carried out, it allows to directly measure the effectiveness of teaching on a semantically deep level: Not only how well a subject is learned, but also which misconceptions were reduced (or caused) by a change in teaching.

An idea for a technology-driven intervention aimed to foster conceptual change which was not integrated into the Backstage 2 ecosystem is developed in [156]: Relevant concepts of a course's topics are represented as interactive concept maps on the platform, which can be created, extended and rearranged by the students. A student who made a systematic error can be asked to rearrange their concept map of the regarding topic. This approach is equally generic as displaying refutation texts (as concept maps can always be drawn), while also being interactive. Note that letting students draw so-called "pre-concept maps" before instruction has been used to assess which misconceptions are present within a population of students [229], and comparing such "pre-concept maps" to "post-concept maps" (drawn after instruction) have been used to assess conceptual change [116].

[2]see Appendix A for descriptions of these courses

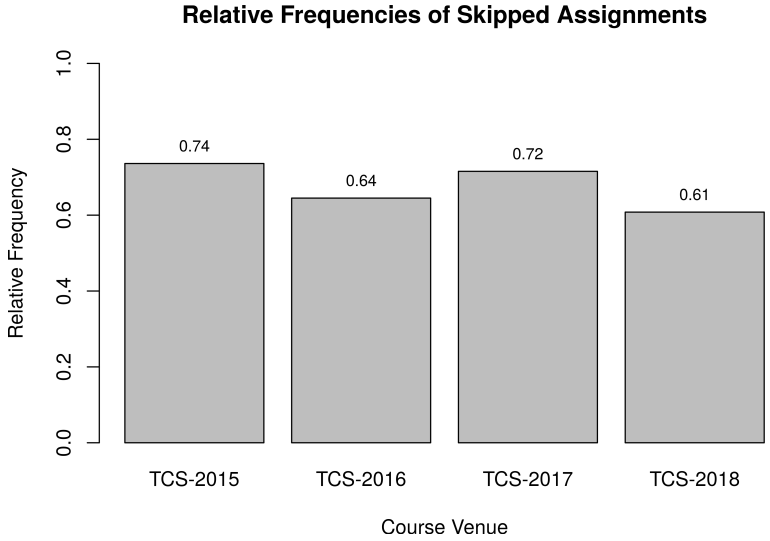


Figure 6.4: Relative frequency of skipped assignments throughout different course venues of theoretical computer science, in 2018 analytics-based nudging was implemented.

6.2 Fostering Behavioural Change

This section describes behavioural changes which were fostered with Backstage 2 / Projects. First, influences of analytics-based nudging (described in Section 4.3) are reported, then a general change in the use of the platform between two courses is described.

The influence of nudging. The learning and teaching format "Analytics-based Nudging" encompasses the provision of predictions of a student's examination fitness (see Section 5.2) and risk of skipping homework (Section 5.1) to nudge students to deliver (more) homework. A first evaluation of this learning and teaching format was conducted in a course on theoretical computer science (TCS-2018). The results of this evaluation are published in [106]. In this course, in which personal analytics were provided in an online dashboard on the platform, students skipped fewer homework submissions than in the three previous course venues, as seen in Figure 6.4. While the homework skipping rate of 2018 was significantly lower to the skipping rates of 2015 and 2017, it was not significantly lower than in 2016.

A second evaluation was performed in a course on logic and discrete mathematics (LDM-2019). For this evaluation, Backstage 2 / Projects sent personalized analytics reports via email to the course's participants, as well as providing an analytics dashboard (both are described in Section 4.3). Yet, despite the analytics-based nudging being more prominent than in the previous evaluation, no significant decrease in homework skipping was found: Skipping rates changed from 96.8% of all homework being skipped to 96.4%.

Both evaluated courses differed not only in the implemented nudging but also in taught subjects, teaching staff and other details. Furthermore, only in the TCS courses, bonus points for delivered homework were awarded, and students reported missing incentives as a cause of not submitting homework (reported in Section 4.3). This might be a major cause for the large difference in homework skipping rates between the TCS courses (between 61% and 74%) and the LDM courses (96.8% and 96.4%).

A culture of dialogue. In the following, data from LDM-2018 and LDM-2019 are evaluated. Between these course venues, the software remained fairly similar, with the only changes being the implementation of the afore-mentioned analytics reports, and an improved com-

	commented feedback	mean HFD length	commented exercises	mean ED length
LDM-2018	1%	2	29%	2.75
LDM-2019	4%	2.4	9%	1.75

Table 6.1: Comparison of discussion lengths in LDM-2018 and LDM-2019, "commented feedback" refers to the frequency in which students commented on feedback they received from a teacher, "HFD length" refers to the total numbers of comments in a homework feedback dialogue, and "ED length" refers to the total numbers of comments in an exercise dialogue.

munication awareness (users could choose to be informed via email on general activities in 2019, while in the previous venue they were only informed when their homework received feedback). Also, the exercise texts of LDM-2019 were improved or reworked by the teaching staff using comments which students left in LDM-2018. As mentioned before, homework delivery rates could not be increased between LDM-2018 and LDM-2019, but other behaviours changed between these course venues.

Firstly, students replied more often to reviews they got from their teachers in LDM-2019 than in LDM-2018, and these "homework-feedback-dialogues" (HFDs) increased significantly in length. In both courses, HFDs were initiated by the student either with a final remark (like "thank you" or "Ok, I see") or with an inquiry asking further questions on the exercise, the given solution, or the feedback. In both courses, "final remarks" and "inquiries" were posted approximately equally often. Notably, in LDM-2018 there were student inquiries which *remained unanswered* by the teachers, and this did not happen in LDM-2019.

Secondly, exercises (i.e. problem statements) received significantly *fewer* comments in LDM-2019 than in LDM-2018, and these exercise dialogues (EDs) were shorter. In both courses, comments posted on exercises only contained questions regarding the exercises. These differences in communication behaviour are displayed in Table 6.1.

Thirdly, students, including those who neither delivered any homework nor left any comments, spent more time on the platform in LDM-2019 than in LDM-2018: In LDM-2018 an average of 6.8 active hours, and in LDM-2019 an average of 14.8 active hours was measured[3].

In summary: While homework delivery rates stayed the same in both courses, the *use* of the platform changed. Students visited the platform more often (possibly to see homework submissions of other students and the feedback they received, as submissions made up the majority of the content available), and engaged more in feedback dialogues with their teachers. At the same time, the course's exercises themselves seemed to have caused fewer questions than in the previous course venue, a possible result of the attempted exercise improvements.

The increase in feedback dialogues and time spent on the platform may have been caused by the improvement of communication awareness on the platform. Furthermore, it has to be noted that the learning management systems used *prior* to Backstage 2 / Projects did neither provide any communication functionalities to publicly discuss feedback or exercises nor the possibility to see the homework submissions of other students. Possibly, students needed time to perceive Backstage 2 / Projects as a social medium and not only as a mere homework delivery platform.

Nicol argues that teacher feedback has to be provided as a dialogue instead of a one-way communication [171]. The results above show that simply providing the students and teachers with the technology to do so (in LDM-2018) is initially not sufficient, but that a

[3] an active hour is defined as an hour in which at least one user-action is registered on the platform.

"culture of dialogue" can emerge when simple awareness measures are implemented and students and teachers alike get used to the new functionalities.

Perspectives. The results regarding analytics-based nudging indicated that students can (in certain cases) be motivated by such nudging, but also that other motivators such as bonus points are a much more effective motivator. Arguably, further developments of analytics-based nudging should reflect how the "nudged behaviour" influences or is influenced by other factors such as the workload in other courses or external motivations.

One perspective for future developments would be to motivate students to become active early within a learning phase (for instance shortly *after* an assignment is given and not shortly *before* the deadline). Firstly, this would foster a spacing effect, which is beneficial for learning [227], and secondly, it could increase the efficiency of exercise dialogues, as it could allow more students to benefit from the answers while working on their homework. Communication on the platform happened solely between teachers and students, never did students enter dialogues with their peers. Yet, such communication could reduce the teachers' workload [172], and future developments of the software could aim to foster such dialogues.

6.3 Sustaining Exploratory Learning

In Section 4.5, the code editor Knoala and its use in enabling exploratory learning of formal languages is described. The evaluation presented in Section 4.5 showed that, while many students appreciated the editor's functionalities and used it for studying, a large number of students struggled with the "syntax barrier" and used Knoala only rarely.

In Section 6.1, results of a laboratory study were reported, which indicated that the learning of unintuitive concepts could be fostered by implementing a simple scaffolding. This study and the implemented improvements are discussed in greater detail in the bachelor thesis of Galina Keil [130]. This section further evaluates the data gathered in that laboratory study with respect to the successful use of the editor and patterns of exploratory learning.

Successful use. As mentioned before, students participating in the laboratory study, learned to understand and use the theoretical programming languages LOOP and WHILE with the help of Knoala. They were divided into a scaffold group (students who used Knoala *with* scaffolding), and a *default* group (students who used Knoala *without* scaffolding).

During the study, students were asked to perform correction tasks (in which they were provided with erroneous source code they should correct) and programming tasks (in which they should write programs realising a given specification). As seen in Figure 6.5 Students in the *scaffold* group completed the tasks significantly more often and in a significantly shorter time than students in the *default* group.

In Section 5.4, the notion of "programming competence levels" was introduced with

- **level 0** being the ability to provide code in the correct format,
- **level 1** being the ability to write syntactically correct code,
- and **level 2** being the ability to write semantically correct code.

The laboratory study required students to use a correctly configured code editor, hence all students reached level 0 by study design. Figure 6.6 shows that, in both groups, more students reached level 0 than level 1, and more students reached level 1 than level 2. This result was to be expected as higher levels entailed more complex tasks. However, it is striking that scaffolded students reached higher competence levels easier.

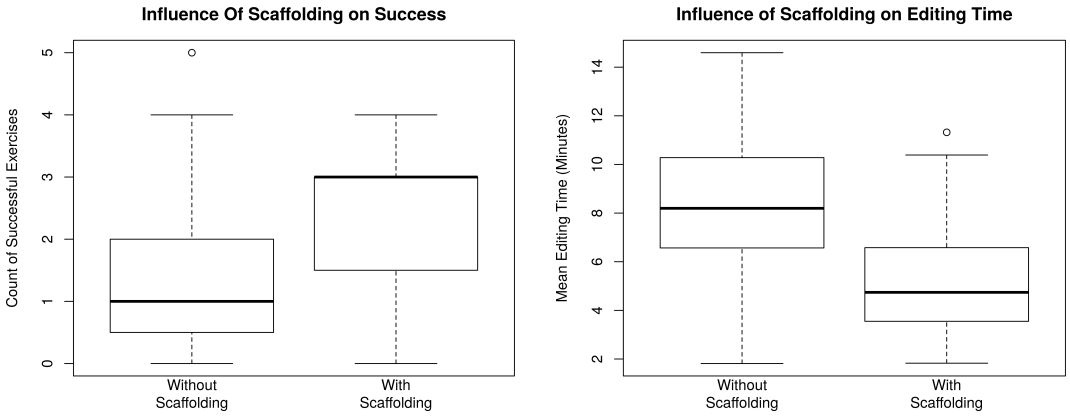


Figure 6.5: Comparisons of behaviour in the *scaffold* and *default* group. Left: Task completion rates. Right: Mean times students spent editing exercise solutions.

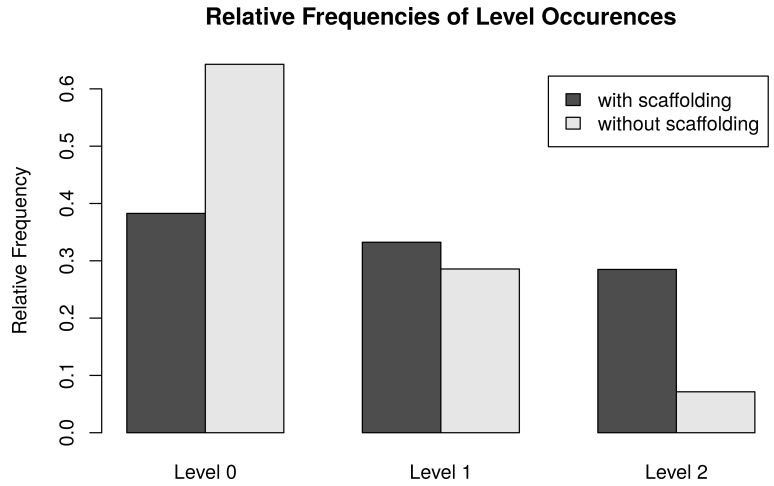


Figure 6.6: Relative frequencies of reached the competence levels (counting the highest reached level for each student and exercise) in the *scaffold* and *default* group.

Flow and play. Besides being more successful in writing correct programs, students in the *scaffold* group used Knoala differently than students in the *default* group. Firstly, scaffolded students *ran* their programs less often while solving exercises than non-scaffolded students (on average 4.5 times per exercise in the *scaffold* group, and 6.8 times in the *default* group). Furthermore, after having run *syntactically correct* code, students in the default group ran *syntactically incorrect* code more often: Syntactically incorrect code followed syntactically correct code in 32% of the cases in the default group, and only 14% of the times in the scaffold group. These differences may have been caused by the continuous display of syntax errors in the scaffold group. Students in the default group had to run their code to verify its syntactic correctness.

Arguably, the laborious process of having to run the source code and of finding syntax errors without further indications broke the workflow for students in the default group, which might explain the increased time these students needed to complete exercises.

To further investigate how the use of Knoala differed in both groups, all code samples executed during the study were categorized in the following way:

1. Two code samples which both are semantically correct belong to the same category.
2. Two code samples which are both *semantically incorrect*, yet *syntactically correct* belong to the same category if they compute the same result (which is incorrect for the task at hand).
3. Two code samples which are *syntactically incorrect* belong to the same category if they were caused by the same syntax error (such as a misplaced semicolon, as described before). Here, two syntax errors were identified to be identical if they produced the same error message disregarding line numbers and variable names.

Evaluating the occurrence frequencies of these categories revealed that "scaffolded" students ran "semantically incorrect" code belonging to categories which *did not occur* in the *default* group. Some of these "scaffold-only" categories contained code which obviously resulted from students trying out variations of the same program. One relatively large category, for instance, entails programs which compute the triple of the input while the exercise asked to compute the double. These categories were often reached *after* having executed semantically correct code: Some students in the scaffold group *played* with their source code after having solved an exercise. This behaviour was not observed in the default group.

The observation of "play" is further illustrated by Figure 6.7, which shows trajectories of categories the scaffolded students' code passed while programming exercises were worked out. The three highlighted clusters encompass closely connected categories, one of which (highlighted in blue) contains only syntactically correct code. Students "passing" this cluster were able to consistently test and manipulate the semantics of their programs without destroying its syntactical correctness. Notably, this cluster was only entered *after* the exercise was solved correctly.

Perspectives. The implemented scaffolding improved Knoala. Yet it has to be noted, that the implemented improvements are not perfect: Among others, the displayed error messages are sometimes ambiguous, and sometimes point one line below the actual error. While further improvements could try to remedy these problems, it can be doubted whether such improvements would have an equally large effect on the students' learning as the scaffolding implemented so far.

It would be interesting to detect "patterns of exploratory learning and play" outside the laboratory. In general, any educational software could be evaluated with regard to whether it allows or encourages such learning and how it enables the detection of such behaviours.

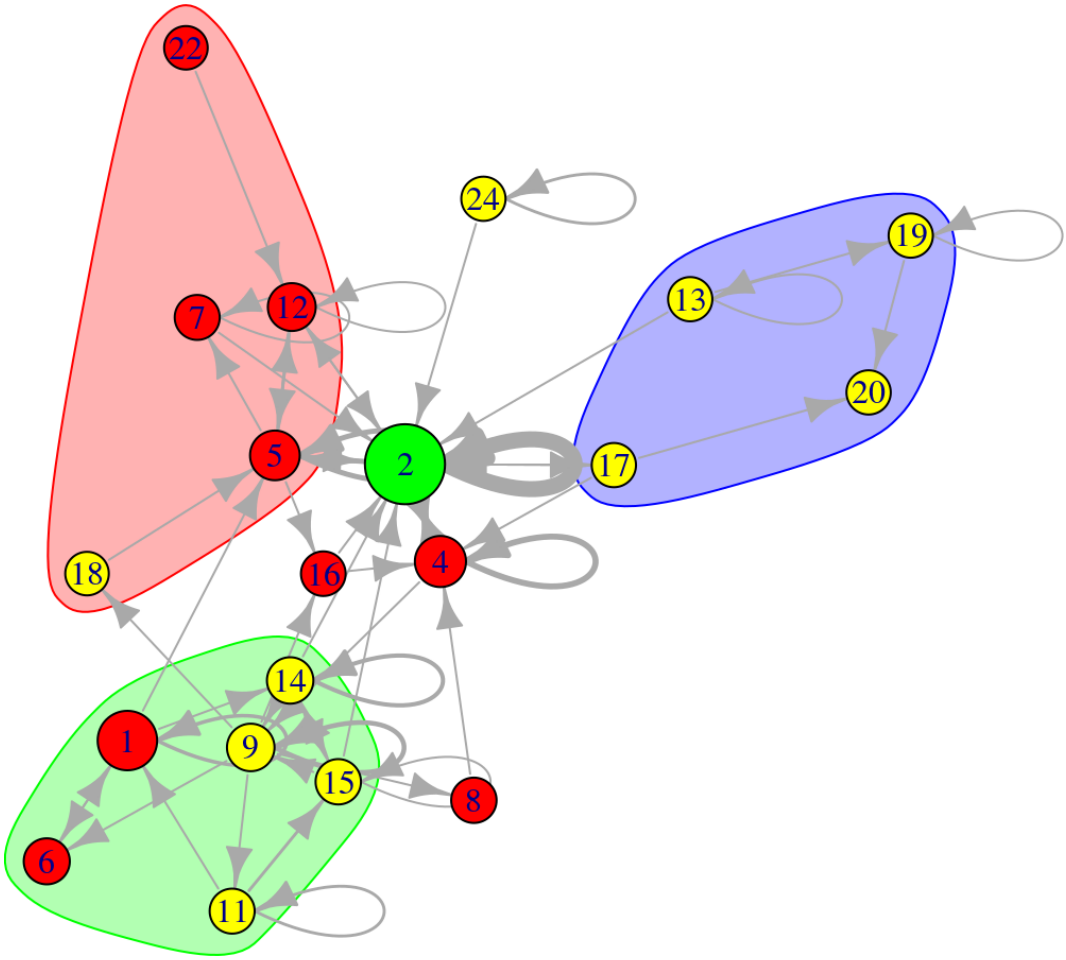


Figure 6.7: Trajectories of code categories in the *scaffold* group. Nodes signify code categories, with size indicating occurrence frequency. Red nodes represent code with syntax errors, yellow nodes represent code with semantic errors, and the green node represents semantically correct code. An edge from node C to node D signifies that at least once a student ran code in category D directly after running code in category C. Only categories which were visited more than twice are shown. Highlighted clusters represent densely connected subgraphs. Descriptions of the error categories can be found in Appendix B.

Note that the detection of playful behaviour was only possible because, besides logging all the code run by the students, the software also logged *the exercise* the students were working on when running the code. Only this allowed to analyze whether a syntactically correct code was semantically correct or to speculate whether a code was a result of "playing around" with a working solution. While in its current form, Knoala does not provide these capacities natively, it would be possible to integrate testing functionalities into the definition of the languages Knoala provides. LOOP-code documents, for instance, could encompass test cases (in a syntax yet to be defined) which the LOOP-code should satisfy, and the LOOP-interpreter could be extended to run the test cases and respond with sensible messages. Such an extension would ease future evaluations.

In the evaluation presented above, errors were detected automatically, leading to 11 distinct syntax errors, and 14 distinct semantic errors which occurred more than twice. One way to further help students in overcoming these errors could be to let teachers write specific help texts, which the software could then display if it detects the regarding errors. Yet, rather than requiring 25 different texts to address each error, teachers could focus on the most common errors or the ones that took the students the longest to resolve. Further, the clusters of errors shown in Figure 6.7 could help in deciding whether an error *actually is* an error, or rather a sign of a playful behaviour (which the software arguably should not interrupt).

6.4 Discussion

This section discusses the previously presented results by first considering general limitations, and then discussing the results concerning self-regulated and teaching efficiency.

Limitations. Most of the results presented above rely on data gathered in courses on theoretical computer science (four course venues) and courses on logic and discrete structures (two course venues). Arguably, this allows reporting relatively certain results *concerning these courses*: While parts of the teaching staff changed, the exercises and topics stayed largely the same. Yet, transferring the results *to other courses* might not be possible. For instance, theoretical computer science might be a field *prone* to cause a large number of systematic errors, or errors that can very well be eradicated using refutation texts. Similarly, students in LDM-2019 might have asked fewer questions regarding the course's exercises because they were higher-achieving students, or simply less concerned with understanding the exercises.

Furthermore, the evaluation does not consider which courses were heard by the students in parallel. As discussed before, certain properties can divert attention between concurrently attended courses. Also, using Backstage 2 / Projects was non-obligatory in all of the examined courses (while teachers encouraged its use), and obviously the results presented here can only reflect the behaviour of learners who chose to use the software.

The results regarding exploratory learning in Section 6.3 were based on a laboratory study. These results might be caused by a biased or undersized sample of participants, and might simply not be transferable to everyday teaching.

Fostering self-regulated learning. According to Zimmerman, self-regulated learners "personally activate, alter, and sustain their learning practices in specific contexts" [267, p. 307], and certain results presented in this report indeed indicate that Backstage 2 / Projects fostered self-regulated learning among students.

Students changed certain misconceptions based on descriptions of systematic errors provided on the platform, which required them to reflect on the conceptions they had and whether these could be true. This kind of meta-cognition is considered a part of self-regulated learning [190].

While these refutation texts seem to have enabled self-regulated learning in the sense of "self-reflection", the results indicate that self-regulated learning in the sense of "self-motivation" was required to benefit from them: Students who were not active on the platform (besides in submitting homework) did not show conceptual change as often as their more active peers. These students were often not motivated to read the error descriptions regarding the homework they attempted or to ask questions if they found them incomprehensible.

As shown in Section 4.5, the "syntax barrier" was a challenge for many students who tried to use the online code editor Knoala. It has to be noted that the students were not required to use Knoala, so *every* attempt to use it can be regarded as self-regulated learning. In Section 6.3 it was shown that implementing scaffolding "lowered" the syntax barrier. It seems likely that these improvements could increase the self-regulated use of Knoala in future, as they also encouraged exploratory "playful" learning.

A problem arguably related to learner self-regulation are decreasing numbers of homework submissions throughout a course (described for instance in Section 4.3). To increase self-regulation in this regard (i.e. motivating students to submit homework) students were provided with personalized learning analytics reports in an analytics dashboard (in TCS-2018) and both in an analytics dashboard and through analytics reports via email (in LDM-2019). The evaluation revealed mixed results: While the intervention seems to have motivated the students in TCS-2018, such a result could not be reproduced in LDM-2019.

Further evaluations revealed that, in LDM-2019, a major reason for not submitting homework was a lack of incentives, or, more precisely, that other courses provided *concrete* incentives (bonus points for the examination), while LDM-2019 provided solely *intangible* incentives in the form of predictions. Arguably, this diverted the students' motivation towards other courses, as an *intangible* prediction "you will probably be 1% better in the examination if you submit this assignment" is simply outperformed by a tangible bonus "you will get 1% on your final examination grade if you submit this assignment". Interestingly, the *combination* of tangible and intangible incentives in TCS-2018 seemed to have motivated more students compared to the previous course venues in which only tangible incentives were offered.

Yet, certain forms of self-regulation increased between LDM-2018 and LDM-2019 without them being intentionally reinforced by the teachers or the software: Students replied more often on the feedback they received on their homework, and visited the platform more often. Possible causes may be the improvement of awareness functionalities between the courses (better notification emails and news-feeds, described in Section 4.1) as well as a changed mindset of the students regarding what the software should and can provide.

Increasing teaching efficiency. Embedding documents containing descriptions of systematic errors in the platform increased teaching effectiveness. Firstly, teachers reported that using these labels for giving feedback decreased their workload (see Section 4.2, and [104]). Secondly, students who were provided with error descriptions (and were active on the platform) avoided the presented errors. While it is unclear whether these students made *fewer errors in total* (they could have come up with new errors to make), the intervention arguably increased teaching effectiveness as it allowed teachers either to find new systematic errors (which they did not) or to focus on the new individual errors students made.

Improving teaching material (such as exercises) is a different way of increasing teaching effectiveness. Indeed, the improved exercises of LDM-2019, which addressed the feedback students gave in LDM-2018, received fewer questions than the same exercises in the previous venue, and consequently teachers spent less time answering questions.

Furthermore, in LDM-2019, students replied more often to the feedback they got on their homework, often asking questions regarding the feedback or their solutions. Allowing

(or even encouraging) such questions does not decrease the teachers' workload. Yet it can nevertheless increase the teaching *effectiveness* as giving feedback the student does not understand (a problem especially prevalent for low-achieving students [178, 195]) is very inefficient, and by encouraging dialogue, feedback can become more comprehensible.

Arguably, Backstage 2 / Projects improved teaching effectiveness most by pointing out *how* teaching could be improved; for instance by displaying systematic errors and student questions on exercises.

CHAPTER 7

Perspectives

This chapter develops perspectives for future research connected to the results presented in this report and to tertiary STEM education in general. First, a broad software-driven approach referred to as "agile learning format development" is defined. In Section 7.1, this approach is applied to the learning and teaching formats introduced in Chapter 4 with respect to the results presented in Chapter 6. Section 7.2 suggests further developments of learning and teaching formats using "prescriptive learning analytics", in which software makes pedagogical decisions based on data gathered within a course.

Agile learning format development. The term "agile software development" was coined in 2001 in a document known (among software developers) as the "agile manifesto" [88]. The manifesto states desirable properties of the process of software development, which, in general, promote flexibility (for instance when changing plans) and communication (between developers, customers, managers, etc.). "Agile", which is often confusingly used as a noun instead of an adjective, has since become a term commonly known among software developers, and providing "Agile coaching" has become a proper industry [74].

Testing is an important component of agile software development, which is reflected by approaches like "continuous integration" [87], "exploratory testing" [12], or "test-driven development" [119]. In these approaches, testing is often one step in an iterative process: It provides means to realize flexibility *during* development, it is not a tool to confirm software qualities *after* development.

Certain approaches presented in the education literature are strikingly similar to the general ideas of agile software development: The exploratory assessment theory [79] and the cognitive apprenticeship model [62] emphasize that learning has to be assessed *during* the learning process, instead of *after* a learning process in order to timely react to arising problems. The eXtreme apprenticeship model [249] (a name obviously referring to the agile software development method "Extreme Programming"[17]) comprises short feedback–work iterations, which is a common feature of agile software development processes. Face-to-face communication between customers and programmers is emphasized in the agile manifesto. Similarly, two-way feedback between students and teachers is emphasized in the literature [171, 172, 38].

While the "agile manifesto" explicitly argued to favour flexibility *over* sticking to fixed processes, the agile software development literature devotes much effort to specifying and evaluating fixed processes (like Scrum [223], Kanban [2], and Extreme Programming [17]). Similarly, much education literature is devoted to defining fixed learning processes in the form of learning and teaching formats [3, 36] (and this report's chapter 4), or collaboration scripts [137, 65, 68]. Arguably, a major reason to advocate for flexibility in software development is the unpredictability of problems which will be encountered *while programming*. Similarly, it is difficult for teachers to foresee what exactly will pose problems to the learners *while learning* (and frequently, the causes of learning problems surprise teachers).

Obviously, software development is different from teaching. Courses are usually taught repeatedly in several venues, while the same software is usually not re-developed on a regular basis. Yet, the process of *improving* teaching can be compared to software developments processes like continuous integration, where working software is steadily improved and repaired. The following (simplistic) algorithm is derived from the iterative processes found in the agile software development literature (for instance [223] or [17]), and aims to flexibly improve technology-driven learning and teaching formats:

1. Identify learning or teaching problems by collecting and evaluating learning analytics.
2. Devise an intervention and, at the same time, define how newly gathered learning analytics would reflect the success of the intervention.
3. Implement the intervention, collect data, and evaluate it with respect to the criteria devised in step 2. Include the intervention in the current teaching format if the intervention was successful, and discard it otherwise. Go to step 1.

In fact, the learning and teaching formats presented in Section 4 are in part a result of a similar process as they are focussed on observed problems (low homework completion rates, problems of students in using abstract formalisms, etc).

While this algorithm aims to improve *learning and teaching formats* (the iteration length being a whole course of 3 months) it can be applied on different scales: Within a lesson (for instance if students have problems which need timely solutions), within an educational institution such as a school or a faculty (for instance if motivating students in certain courses diverts attention from other courses), or even within governmental structures (for instance if there is a considerable impact of the socio-economic status on educational achievement as reported in [58]).

Educational software is the key component for such improvements because it allows collecting and evaluating learning analytics for a large number of students. Hattie writes "It was only when I discovered that feedback was most powerful when it is from the student to the teacher that I started to understand it better" [101, chap. 9]. Obviously, Hattie regards feedback given by students as a means to (flexibly) improve teaching. Similar to Hattie's revelation, one could argue that learning analytics are most effective if provided *to the teachers*. "Agile learning format development" is a proposal for an approach to developing and adapting teaching formats by presenting teachers with automatically generated feedback.

7.1 Improving Learning and Teaching Formats

This section presents perspectives for improving the learning and teaching formats conducted with Backstage 2 / Projects by following the approach defined above: Reporting learning or teaching problems which have been observed, suggesting interventions and how intervention success could be measured. The focus hereby lies on further developments of

learning and teaching formats discussed in Section 4. Note that some problems reported in the following are only supported by anecdotal evidence. In these cases, the focus lies on developing appropriate measurements to concretely detect the problem.

Often, implementing the suggested interventions, as well as setting up the needed structures for data collection, would require extending the software. Suggestions are made on how to re-use as much of the existing infrastructures as possible.

The gap between synchronous and asynchronous learning. The Backstage 2 ecosystem encompasses, besides the "project component" for asynchronous learning, a "course component" for synchronous learning. This component provides a digital backchannel in form of a collaborative annotation system and an audience response system [155] to its users. Two courses on logic and discrete mathematics (LDM-2018 and LDM-2019) were supported with both synchronous services (for lecture slides, collaborative annotations, and lecture-accompanying quizzes) and asynchronous services (analytics-based nudging and homework delivery).

In general, we registered much more activity in the synchronous component than in the asynchronous component. While the homework delivery rates lay in both courses between 4% and 5%, about 30% of the students joined the digital backchannel during lectures, and lecture quizzes (multiple choice quizzes which were interleaved with the lecture slides) were played by 70% of students who were online [155].

This observation obviously points to a problem, as *all* components of a course are considered helpful by the teachers to successfully learn the subject.

There may be different causes for the observed "gap" in activity. Firstly, synchronous activities require less self-regulation from the students than asynchronous activities. Playing multiple-choice quizzes takes minutes, while working out exercise solutions can take (depending on the exercise) hours or days. Also, playing a quiz consists in *choosing* an answer, while homework usually consists in *constructing* a solution from a blank slate. Secondly, students might be tempted to think that quiz questions are a better preparation for the examination than elaborate exercises. This argument is not easily refutable as, on the one hand, examinations often contain multiple choice quizzes, and, on the other hand, homework exercises often require solutions which could not possibly be constructed within the limited time of an examination.

Arguably, a further investigation of possible causes would require surveys or interviews with the students, yet a first intervention would aim to better integrate of the two learning contexts. This could be achieved by allowing artefacts to be *transferred between contexts*. An example of a transfer from a synchronous to an asynchronous context would be if a teacher starts to solve an example problem in class, and all students are tasked to complete the partial solution as homework. Discussions started in the synchronous class context could carry over to the asynchronous homework context. This could reduce the hurdle to get started, as the teacher could (flexibly) react to the first problems or objections expressed by the students.

An example for a transfer from an asynchronous to a synchronous context could be realized by flexibly scheduling discussions of specific topics to specific lessons. If for instance a systematic error is very prevalent (or the teacher deems its discussion important), it could be scheduled to be discussed on the next occasion. This could convey the importance of learning outside the classroom: If the teacher mentions something in a lecture, it must be important (and so must be asynchronous learning).

These new functionalities would require relatively few extensions of the Backstage 2 ecosystem, as documents (the artefacts created in a learning context) are stored by a central service

every component can use. Hence, providing such functionalities would solely require to devise a suited user interface (which has nevertheless to be carefully devised). Yet, *using* the new functionalities efficiently would require effort from the teachers: For instance, teachers would have to be willing (and able) to devote a part of their lecture time to a topic which was scheduled by the software.

The metric for success of these interventions is obvious: A more balanced use of both synchronous and asynchronous services by the students (without the use being generally lower). The long term goal would be to encourage students to start their own synchronous and asynchronous contexts (for instance to organize learner groups or learning projects), and to let *them* regulate the transfer between *their* contexts.

Fostering intrinsic motivation. Obviously, examination results are a main motivator for students, and a number of observations reported in the previous chapters sustain this claim: Consider for instance motivations for homework delivery reported by the students which are discussed in Section 6.2, or the fact that a considerable number of students started to use the platforms functionalities only days before the examination (Section 4.5).

"Grade motivation" and its numerous detrimental effects on learning has been studied for decades (summarized for instance by Kohn in [138]), with the main effects being a diminished interest in the learned subject, an increased preference for the easiest possible task, and a reduction of the students' quality of thinking (manifested through behaviours like skimming books for "what they I need to know") [138]. Arguably, abolishing grades completely (as Kohn requests) is an endeavour outside the scope of this research. Yet, if not abolishing grades, educational software could help in shifting the focus away from grades towards the actual learning content. Interventions with this goal could try to foster playful exploratory learning as illustrated in Section 6.3, or point out concrete learning problems or learner achievements. The representation of commonly held misconceptions or common errors as discussed in Section 4.2 is certainly a step in this direction, and further interventions could emphasize this feature more (for instance through synchronous class discussions as mentioned above). An idea to reify student achievements within the Backstage 2 ecosystem is given in [156]: Concrete achievements of a student are represented by tokens of various playful forms (for instance trees blossoming or withering depending on the completion of a task).

Arguably, nudging students with predictions of examination outcomes as discussed in Section 4.3 might have increased their grade motivation. A perspective for future work would be to assess whether this is indeed an effect of such nudging.

These interventions would be known to work if playful behaviour was observed (or such behaviour increased), or if the number of students only visiting the platform shortly before the examination decreased.

Flexibly-flipped classrooms. Many of the courses supported with learning teaching formats described in Section 4 encompassed lectures (in which a professor conveyed new content) and practical lessons (in which homework exercises were discussed). Certain tutors, including the author of this report, frequently had the impression that a substantial number of students visited the practical lessons simply to "copy the board" where exemplary solutions were presented, instead of actively engaging with the problems (not to mention presenting alternative solutions, or posing further questions). Obviously, teaching a group of students consisting mostly of "passive copiers" is unpleasant, and ineffective: Copying material is not learning, and therefore presenting anything solely for it to be copied is not teaching.

A flipped classroom could remedy such effects, as it requires student activity. Also, topics discussed in the practice lesson could be scheduled flexibly to the students' needs, who

could be asked to prepare questions they have on the exercises they are currently attempting. Students could endorse questions (e.g. through an upvote mechanism), which could lead to the questions being discussed in more (possibly concurrently held) lecture sessions. Additionally, students could be provided with exemplary solutions of exercises they already submitted, and questions on these exemplary solutions could be similarly scheduled for discussion.

While the observation of passively solution-copying students is anecdotal, it fits the symptoms of grade motivation, which "tends to diminish students' interest in whatever they are learning" [138]. Measuring this behaviour (probably requiring interviews or surveys) would be a first step, as it would allow to measure the effectiveness of further interventions.

Exploring STEM's formalisms. Backstage 2 / Project's code editor Knoala allows students to explore a number of text-based formal languages. Improvements in the way Knoala reports and displays syntax errors, evaluated in Section 6.3, were shown to lower the syntax barrier for two evaluated formal languages. An extension to Knoala (also proposed in Section 6.3) would be to use block representations of programs (famously introduced by the scratch project [158]) as it could further lower the syntax barrier; one cannot misplace a semicolon, if the blocks don't allow such a placement.

More generally, one can ask which representation (for instance blocks or text) of a formal artefact entails which didactic merits or problems. Arguably, the choice of such a representation can be subtle. Consider for instance the two "proof-editors" presented in Figure 7.1. Both editors allow constructing proofs by natural deduction, and both represent proofs as trees. Yet, in the right one (which is integrated in Backstage 2[1]), the depth of an expression reflects the number of steps needed to prove that expression. In the left one, the depth of an expression reflects the number of assumptions which have to be made to prove that expression. Both representations might have specific *didactic* merits, while being technically equivalent. Similar "concurrent" representations can be found throughout the STEM fields: Turing machines can represented by finite automata or by operation code, algorithms can be represented by flow-charts or by source code, molecules can be represented by structural or molecular formulas, etc. Indeed, it is part of each STEM profession to choose a representation suited for the current problem.

A considerable perspective regarding exploratory learning of formal languages would be to allow learners to switch seamlessly between such different representations, or to show *several* representations simultaneously and continuously reflect changes made to either of them.

A different kind of extension could allow students to interactively trace the steps an interpreter takes in executing or verifying an artefact. Debugger interfaces, which are commonly integrated in software development environments, enable such interactions for many programming languages. Such interfaces allow the user to stop the execution of a program at any point. The user can then examine the state of the runtime environment (e.g. which variables are bound to which values) and let the program run stepwise or let the interpreter continue the execution of the program. Undoubtedly, this is not only a powerful tool to find bugs in a program, but also to explore *how* a program is evaluated. Such an interface could be implemented for the execution of Turing machines, just like for the verification of formal proofs.

Collecting usage data of the new features would then give insight on whether more students are successful in using STEM's formal languages, and whether more students are encouraged to explore the formalisms on their own initiatives.

[1]This editor was implemented by Korbinian Staudacher for his bachelor thesis which is found in [234].

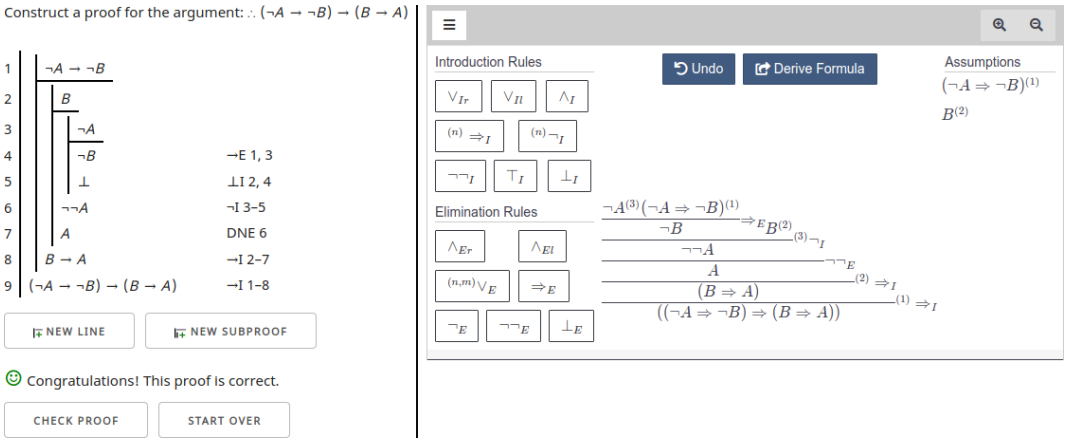


Figure 7.1: Two proof editors for prooffs by natural deduction. Both editors contain correct proofs of the general validity of the formula $(\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A)$. The left editor is available at `https://proofs.openlogicproject.org/` (retrieved 29th of January 2020), and follows a syntax described in [157]. The right editor is integrated in Backstage 2 and is described in [235]

.

7.2 Towards Prescriptive Learning Analytics

Prescriptive Learning Analytics refers to educational software which makes didactic decisions based on learning analytics [259]. At the time of writing this report, such applications are rare, as learning analytics are mostly used for assessment or monitoring [11] (see Section 3.1 for a brief discussion of the literature).

Analytics-based nudging as introduced in Section 4.3 can be, to a certain degree, considered as "prescriptive" because the personal analytics sent to students include personalized "nudging paragraphs". Obviously, with learning analytics available in a learning platform, more sophisticated "prescriptions" are possible, and this section tries to give perspectives on such applications.

Scripted peer teaching. In Section 4.4, a teaching format relying on peer teaching was introduced. The evaluation in a course on functional programming showed that a main problem of the format was a lack of participation from the students, who often did not provide the peer reviews they were tasked with. At the same time, the students who *did participate* the most, and who could have arguably sustained the format if there were no rarely participating students, were also the most *proficient* students. Proficiency was hereby assessed as the ability to write semantically correct source code.

In a follow-up study, a second instance of the course format was conducted in a course on logic and discrete mathematics (between LDM-2018 and LDM-2019). The course required the students to deliver homework using a number of mathematical formalisms which were supported by the platform. Such homework submissions were automatically assessed in a similar manner as in the course on functional programming.

Predictions on the students' abilities to write semantically correct "code" were then computed using the predictor described in Section 5.4. These predictions were then used to intelligently pair students so that the most proficient student would review, and be reviewed by, the least proficient students. The exact mechanism for this pairing is described Robert Pospisil's master thesis in [193].

This approach has similarities to the "calibrated peer review" process presented in [194]. Calibrated peer review requires students to perform tasks in a "calibration step" in which their ability to make correct peer-assessments is measured by comparing their assessments against teacher-given assessments. The peer teaching approach presented here simply uses the student's proficiency *in the taught subject* instead of "calibration tasks" to approximate their assessment capabilities. Yet, the further *uses* of these approximations differ: Calibrated peer review is centred at *peer grading*: Grades are calculated as a weighted averages of all peer-assessed grades with weights reflecting the assessment capabilities of the students. In the approach presented here, the focus lies on *peer teaching* and estimated assessment capabilities are used to maximize feedback efficiency, as arguably the least proficient students could benefit the most from the best feedback.

Sadly, and despite the elaborated Learning Analytics orchestration machinery in place, the experiment failed. The number of peer reviews given by the 38 participants was even lower than in the first study. Arguably, this was not entirely the fault of the altered pairing mechanism, as the peer-review participation was low from the start (before sensible assessments could be made). Besides from treating a different subject than the course in the first study, certain course arrangements were made differently for the follow-up study. For one, this course encompassed presence sessions every three weeks, which may have caused the students not to care too much about peer teaching, as problems could be discussed with a teacher face-to-face. Also, as only very few peer reviews were given, the tutors started (thankfully) to give feedback on the submissions after the peer review phase had ended. This is a sensible decision (students often needed the feedback), but this may have decreased the motivation for giving peer reviews even more: Why bothering with giving peer reviews if the teacher will provide feedback anyway?

Spontaneous peer teaching. Apart from using concrete assignments to realize peer teaching (which entails certain problems as seen above), one could try to sustain *spontaneous* peer teaching between students. As a starting point for such a communication a "help-button" was added to the assignment interface. This feature became available in the second half of the course LDM-2019. By pressing on a button labelled "Help me, I'm stuck!" a two-step dialogue was opened. In a first step, the students were provided with the current content of the assignment's project (arranged in categories like "currently discussed", or "current questions"). This step was introduced to avoid double questions, as a student having a specific problem might not be the first to have that problem. If the student was not satisfied with the content available, they could choose to send a help request. In this second step, the student was asked to specify their questions. The help request was then forwarded to the teachers of the course and two students of the course having the highest predicted examination outcome (estimated by an examination fitness predictor described in Section 5.2). Yet, in order to be able to send a help request, the student had to first provide an attempt of a solution, a restriction implemented to avoid the platform being flooded with requests simply asking for solutions instead of help.

Sadly, this feature was only rarely used and did (even more sadly) not work properly: Only two students (of about 500) send help requests, which were, due to a software error, not forwarded to their peers, but only to the course's teachers.

Nevertheless, the data gained from this experiment allows certain insights. The two students who posted a help request each, were (arguably not coincidently) the two most proficient students in the course (as estimated by the examination fitness predictor). Notably, their questions were well formulated and precise. Furthermore, 12 other students reached step 1 of the two-step process and aborted there. In total, 17 help requests were aborted in that manner. This might be a sign that step one actually helped the students because they were satisfied with the content they were presented with, yet, in most of the cases there was

simply no additional content available apart from the exercise texts. More likely, students were discouraged when seeing that they had to provide a first attempt in order to get help.

This explanation is underpinned by the observed *timing* of interactions. None of the students who only reached step one saved solution attempts *before* querying the system for help. Yet, they frequently queried the system for helpful content and abandoned their requests after a few minutes or hours. The two students who actually posted help requests did show a similar behaviour, frequently querying the system, and spend 42 and 51 minutes on the platform *before* finally formulating their questions.

It is important to mention that the two-step process, and the requirement of "having to make an attempt before asking for help", were introduced to counter *anticipated* and not *observed* problems: Double help requests referring to the same problem, and a flood of help requests which simply asked for a solution. In retrospect, this was unfortunate, because the fact that these problems did indeed not occur does not allow the conclusion that the measures did reach their goals. Instead, the data suggests that the implemented measures discouraged the use of the system as a whole.

By referring to the approach developed at the beginning of this chapter, the more "agile" approach would have been to implement a less restrictive help mechanism, letting the students use it, monitor the system for problems, and implementing restrictions if necessary *as a reaction* to these problems (possibly deactivating the service completely if the system would have been flooded by nonsensical requests).

Nevertheless, perspectives for future developments can be drawn from this (failed) experiment. For instance, the system could provide a "Help me, I don't know how to start" button if no solution attempt has been made for the assignment. As about half of the students who pressed the help-button did not submit any homework afterwards, not knowing *how to start* might be a major problem. Indeed, "inert knowledge" [206] (i.e. the inability to apply knowledge to solve a problem), is a phenomenon common for STEM students [204].

Adaptive content choices. Learning analytics can be used to adaptively choose content for a student, and throughout this research, a number of concrete starting points for such adaptive components were suggested. Note that recommender systems are a common application of learning analytics [41], yet usually such applications try to suggest whole courses a learner could join, based on the courses a learner already completed. A different use of adaptive content choices would be to provide a learner with the material they would profit most from to reach *a specific learning goal*.

Such an approach is proposed in Section 5.3: Providing a student with descriptions of systematic errors they might most probably make. To efficiently use such a system, some kind of ontology would be necessary, modelling which errors occur in which kind of exercises, which topics were involved, etc. To this end, the notion of a project based learning platform, where sharing documents between projects seems cumbersome, might not be appropriate. A learning platform supporting such an ontology could realized as a semantic wiki [220]: A wiki with an underlying knowledge structure which allows to draw connections between pages (in this case containing errors, exercises, learning material, and so on).

A perspective for an adaptive choice of content *representation* is given in Section 5.4: Choosing for instance a block or a text interface for coding exercises based on the likelihood of a student to struggle with the syntax of a formalism. A similar approach is commonly used in intelligent tutor systems, which choose exercises fitting the current estimated proficiency level of a learner [41].

CHAPTER 8

Conclusion

In this report, the social learning platform Backstage 2 / Projects and its components were introduced. The platform uses projects (defined as collections of users, documents and assignments) as basic building blocks to define learning and teaching formats. These learning and teaching formats are devised to tackle concrete problems observed in tertiary mass STEM education: Easing the time-consuming work of revising student homework (Section 4.2), nudging students with learning analytics to deliver (more) homework (Section 4.3), using peer teaching to reduce the teachers' workload (Section 4.4), and supporting the learning of STEM's formalisms with automated feedback (Section 4.5).

The software is able to automatically compute learning analytics, which predict skipping and absenteeism (Section 5.1), examination performances (Section 5.2), systematic errors and misconceptions (Section 5.3), and programming competence (Section 5.4). The predictors are designed to be *intelligible* (it is clear how the data reflects the students' learning), *actionable* (the predictions reflect changes in behaviour) and *accurate*. As a main data source, the predictors use teacher-labelled or automatically labelled homework submissions. Collecting labels given by teachers on homework submissions was integrated into teaching in a process similar to grading, and teachers reported a decreased workload when using labels referring to systematic errors.

The learning and teaching formats were evaluated in several case studies focussing on the students perceptions of the software (reported in Section 4) and on the formats' successes in improving learning (reported in Section 6). The students' attitudes were generally positive, with the software's compile and edit functions for formalisms and formal languages being among the best-liked features. Presenting students with predictive learning analytics, predicting examination performances and risks of skipping homework, attracted mixed attitudes. Conceptual change could be fostered by prompting students with descriptions of systematic errors which were collaboratively collected by the teachers of a previous course venue. Behavioural change in the form of motivating students to deliver (more) homework could not consistently be induced: Significant changes in homework delivery rates could be measured between courses on theoretical computer science, but not on logic and discrete mathematics.

Students used, and benefited from, the software's edit and compile functionalities which supported several of the taught formalisms and formal languages. Yet, the majority of students who used these functionalities used them only rarely. Characteristics of the successful use of these functionalities were identified, and are reported in Section 4.5. Problems in mastering the syntax of a taught formalism (the so-called "syntax barrier") were identified to be a major hurdle for the students. Further evaluations (reported in Section 6.3) showed that improving the error messaging of the software's editor could both improve learning of a formal language and foster exploratory learning.

Perspectives for future research were developed, focussing on both the development of learning and teaching formats in tertiary STEM education (Section 7.1) and further developments in the use of learning analytics (Section 7.2). To improve learning and teaching formats, a scheme referred to as "agile learning format development" was devised. Perspectives on the use of learning analytics in tertiary STEM education encompassed "prescriptive" uses of learning analytics which could realised by extending (if not completely relying on) the existing software.

CHAPTER 9

Appendix

A: Table of Evaluated Courses

The following list describes all courses which were evaluated during this research and are referenced throughout this report.

Notation of course periods. "winter" and "summer" stand for winter semester starting in April and ending in August, and winter semesters starting in October and ending in February.

Courses on Functional Programming

These course venues introduced bachelor degree students to programming using the functional programming language Haskell. Topics included, among others: Datatypes, functions and recursion, evaluation, termination properties of programs, and monads.

shorthand:	FP-2017
period:	Summer 2017
number of students:	593
evaluation purpose:	Reference data for FP-2017-p
software use:	None
teaching staff:	One professor and 10 tutors

shorthand:	FP-2017-p
period:	Winter 2017
number of students:	45
evaluation purpose:	Measuring efficacy of peer teaching with minimal teacher intervention for learning.
software use:	Homework delivery and peer teaching as described in Section 4.4
teaching staff:	none

Courses on Theoretical Computer Science

These course venues introduced bachelor degree students to theoretical computer science. Topics included among others: The Chomsky-Hierarchy, finite automata, pushdown automata, Turing-Machines, and complexity classes.

shorthand:	TCS-2015
period:	Summer 2015
number of students:	272
evaluation purpose:	Initial identification of systematic errors
software use:	None
teaching staff:	One professor and four tutors (including the author)

shorthand:	TCS-2016
period:	Summer 2016
number of students:	344
evaluation purpose:	Reference data for TCS-2018, validating occurrence rates of systematic errors.
software use:	Projects were used to provide additional material.
teaching staff:	One professor and three tutors (including the author)

shorthand:	TCS-2017
period:	Summer 2017
number of students:	383
evaluation purpose:	Reference data for TCS-2018, validating occurrence rates of systematic errors
software use:	Projects were used to organize homework and to provide additional material.
teaching staff:	One professor and 5 tutors (including the author)

shorthand:	TCS-2018
period:	Summer 2018
number of students:	338
evaluation purpose:	Measuring the influence of analytics-based nudging on retention and of providing systematic error descriptions on learning.
software use:	Teacher collaboration on written feedback as described in Section 4.2 and analytics-based nudging as described in Section 4.3.
teaching staff:	one professor and 5 tutors (including the author)

Courses on Logic and Discrete Mathematics

These course venues introduced bachelor degree students to formal logic and discrete mathematics. Topics in formal logic included among others: Propositional logic, first order predicate logic, proofs by natural deduction, and proofs by resolution. Topics in discrete mathematics included among others: Proofs by induction, combinatorics, and modular arithmetic.

shorthand:	LDM-2018
period:	Summer 2018
number of students:	614
evaluation purpose:	Reference data for LDM-2019
software use:	Projects were used to organize homework and to provide additional material.
teaching staff:	One professor and 5 tutors

shorthand:	LDM-2019
period:	Summer 2019
number of students:	609
evaluation purpose:	Measuring the influence of analytics-based nudging on retention
software use:	Teacher collaboration on written feedback as described in Section 4.2 and analytics-based nudging as described in Section 4.3.
teaching staff:	one professor and 6 tutors

B: Error Categories of LOOP and WHILE Programs

ID	Level	Description
1	0	expected ; or END, got something else.
2	2	semantically correct.
3	1	no semantics needed (correction task).
4	0	expected variable like xn, got something else.
5	0	expected ; or end of input, got something else.
6	0	expected LOOP, WHILE or variable like xn, got something else.
7	0	incorrect constant declaration.
8	0	incorrect assignment.
9	1	expected 76 got 1219.
10	0	expected DO got something else.
11	1	program get trapped in an infinite loop.
12	0	expected LOOP or variable like xn, got something else.
13	1	expected 26, got 0.
14	1	expected 76, got 24.
15	1	expected 76 got 54.
16	0	expected :=, got something else.
17	1	expected 26, got 13.
18	1	expected 76 got 0.
19	1	expected 26 got 169.
20	1	expected 26 got 14.
21	0	expected LOOP, WHILE, end of input, or variable like xn, got something else.
22	0	expected LOOP, end of input, or variable like xn, got something else.
23	1	expected 76, got 3.
24	1	expected 26, got 28.

C: Surveys

This section lists the questionnaires used for this research.

Demographic Data

This survey was always included.

question: Which is your gender?
response type: male | female | divers | no response

question: How old are you?
response type: Number

question: Name your field(s) of study
response type: Text

question: Name your highest educational degree
response type: Text

question: Which semester are you in? (Referring to your current course of study)
response type: Number

Peer Teaching and Attitudes Towards Peer Teaching

This was administered after FP-2017-p. Results are reported in Section 4.4.

Time management

question: In the course, I spent time on solving assignments
response type: Not at all | less than 2 hours per week |2-5 hours per week |5 hours - 1 day per week |1 day - 2 days per week | more than 2 days per week

question: In the course, I spent time on giving peer review
response type: Not at all | less than 2 hours per week |2-5 hours per week |5 hours - 1 day per week |1 day - 2 days per week | more than 2 days per week

question: In the course, I spent time on reviewing course material
response type: Not at all | less than 2 hours per week |2-5 hours per week |5 hours - 1 day per week |1 day - 2 days per week | more than 2 days per week

question: In the course, I spent time on discussing questions with peers or posting questions
response type: Not at all | less than 2 hours per week |2-5 hours per week |5 hours - 1 day per week |1 day - 2 days per week | more than 2 days per week

General questions.

question: Do you have any additional comments regarding the course or the exercises?
response type: Text

question: I finished the course (I reached topic 10-Monad)
response type: Yes | No

question: Why did you not finish the course?
response type: Text
remark: Only asked if the respondent did indicate not to have finished the course.

Material and software.

question: The course material (apart from the exercises) provided on Backstage was helpful for my examination preparation
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: Compiling and running code documents *provided by others* on Backstage was easy.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: Compiling and running *my own* code documents on Backstage was easy.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: For my learning, using the provided unit tests (typically in Test.hs) was helpful
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: For my learning, using the online compiler was helpful
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

Peer teaching.

question: What are for you the *best aspects* of peer review as conducted in this course?
response type: Text

question: What are for you notable *negative aspects* of peer review as conducted in this course?
response type: Text

question: What are for you notable differences of peer reviews you received and reviews given by tutors in other courses? (If any)
response type: Text

question: Reviewing submissions of others was helpful for my learning.
response type: never | sometimes | most of the time | always

question: Reviewing submissions of others gave me new ideas about possible approaches.
response type: never | sometimes | most of the time | always

question: Reviewing submissions of others allowed me to compare my standard of work.
response type: never | sometimes | most of the time | always

question: Reviewing submissions of others was helpful for my learning.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: Reviewing submissions of others gave me new ideas about possible approaches.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: Reviewing submissions of others allowed me to compare my standard of work.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The reviews I recevived from others were helpful for my learning.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The reviews I recevived from others identified problems in my submission correctly.
response type: never | sometimes | most of the time | always

question: The reviews I recevived from others identified correct solutions correctly.
response type: never | sometimes | most of the time | always

question: The reviews I recevived from others varied in quality.
response type: never | sometimes | most of the time | always

question: The feedback I provided in peer reviews was helpful.
response type: never | sometimes | most of the time | always

question: Do you have any additional comments or ideas on how to improve the peer review process?
response type: Text

Attitudes Towards Analytics-based Nudging

This survey was administered after TCS-2018, and additional questions were administered after LDM-2019. The results of this survey are reported in Section 4.3, and in Section 6.2.

Analytics and Predictions.

question: Have you seen your personal *examination fitness prediction* on Backstage? It might have looked something like this: `image of personal analytics interface`
response type: Yes | No

question: Have you seen your personal *skipping prediction* on Backstage? It might have looked something like this: `image of personal analytics interface`
response type: Yes | No

question: The examination fitness prediction motivated me to learn more
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The examination fitness prediction was interesting for me
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The examination fitness prediction discouraged me
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The examination fitness prediction was helpful
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The examination fitness prediction motivated me to hand in the next assignments
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The skipping prediction motivated me to learn more
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The skipping prediction was interesting for me
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The skipping prediction discouraged me
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The skipping prediction was helpful
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: The skipping prediction motivated me to hand in the next assignments
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

Additional questions for LDM-2019.

question: For how many weeks (approximately) did you hand in assignments via Backstage?
response type: Number

question: If you skipped homework, why?
response type: Text

question: I received analytics reports via e-mail.
response type: Yes | No

question: The analytics reports motivated me to learn more.
response type: Yes | No

question: The analytics reports discouraged me.
response type: Yes | No

question: The analytics reports motivated me to hand in the next assignments.
response type: Yes | No

question: Do you have any remarks regarding the analytics reports?
response type: Text

Attitudes Towards Labels

This survey was administered after TCS-2018. Results are reported in Section 4.2.

Student attitudes.

question: At least once, I read label documents before handing in my assignments.
response type: Yes | No

question: I have seen and read label documents in the course content, or received one as feedback attached to my homework.
response type: Yes | No

question: Knowing the labels for the current exercises was helpful for me.
response type: 6 Point Likert scale ranging from “not at all” to “absolutely”

question: In a future course, if I made a mistake in an exercise, knowing whether (and how many) others made the same mistake would be interesting to me.
response type: 6 Point Likert scale ranging from “not at all” to “absolutely”

question: For my learning, reading a label document attached to my homework as feedback was helpful.
response type: 6 Point Likert scale ranging from “not at all” to “absolutely”

question: Compared to a personal comment, an error description is ...
response type: ... less helpful | ... equally helpful | ... more helpful

question: Do you have further suggestions for the use of labels?
response type: Text

Teacher attitudes.

question: While giving feedback to students, the labels were easy to use.
response type: 6 Point Likert scale ranging from “not at all” to “absolutely”

question: While giving feedback to students, the labels were helpful.
response type: 6 Point Likert scale ranging from “not at all” to “absolutely”

question: While giving feedback to students, the labels were hindering.
response type: 6 Point Likert scale ranging from “not at all” to “absolutely”

question: While giving feedback to students, the labels reduced workload.
response type: 6 Point Likert scale ranging from “not at all” to “absolutely”

question: When giving feedback to students in future courses, I would like to use a software that supports labels.
response type: 6 Point Likert scale ranging from “not at all” to “absolutely”

question: When giving feedback to students in future courses, I would like to share my labels with other tutors.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: When giving feedback to students in future courses, I would like to know how many students received which labels.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: Do you have further suggestions for the use of labels?
response type: Text

Attitudes Towards Knoala

This survey was administered after TCS-2019. Results are reported in Section 4.5.

question: Backstage supports several programming languages and simulators. (For example a Turing machine simulator). I have used a language or simulator on Backstage at least once
response type: Yes | No

question: For my learning, using the online compiler was helpful.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: Creating and running my own code examples was easy.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: Running code examples provided on Backstage was.
response type: 6 Point Likert scale ranging from "not at all" to "absolutely"

question: I used the following languages or simulators
response type: Select an arbritary number of LOOP / WHILE, TuringMachine, GOTO, Pushdown Automaton, Haskell, μ-recursion, CNF Solver (Klauselnormalform editor), Others.

D: Own Publications

General information. Throughout the thesis, all *direct citations* are declared as such. This thesis does not contain direct citations of the author's publications.

Figures taken directly or adapted from other publications (including the author's) are declared as such in the figures' captions.

Often, parts of results presented in one of the author's publications are summarized in this thesis, (e.g. by omitting the methods sections). Perspectives generally extend upon previously published work.

Publication contributions. In the following, the "main contributor" of a publication is the author who contributed most to the planning, conduction, and evaluation of the research as well as to the written report.

- In [103, 104, 105, 106, 107, 108] Niels Heller is the main contributor.
- In [109, 110] contributions were provided in equal parts by Sebastian Mader and Niels Heller
- In [156], Sebastian Mader is the main contributor.

Citations of the author's publications by paragraph. The following list names passages in which the author's publications are cited and states the nature of the citation. Novel contributions (only published in the thesis) are not listed.

- Chapter 4: Technology-Enhanced Formats, *page 19*
  - paragraph **Teaching formats built from components**, *page 21*

    Summarizes an idea presented in [110].
  - Section 4.1: **Software Components and Functionalities**, *page 21*

    Extends work presented in [103, 104, 105, 106, 107, 108].
  - Section 4.2: Teacher Collaboration on Written Feedback, *page 28*
    - paragraph **Preliminary results: Labelling validity**, *page 29*

      Summarizes parts of the results presented in [103].
    - paragraph **Case study results: Attitudes.**, *page 30*

      Summarizes parts of the results presented in [104].
  - Section 4.3: Analytics-based Nudging, *page 32*
    - paragraph **Results: Attitudes towards predictive analytics**, *page 35*

      Summarizes parts of the results presented in [106].
  - Section 4.4: Peer Teaching, *page 38*
    - paragraph **Results: Participation and student competence**, *page 39*

      Summarizes parts of the results presented in [108].
    - paragraph **Results: Attitudes**, *page 41*

      Summarizes parts of the results presented in [108].
  - Section 4.5: Exploratory Learning of Formal Languanges, *page 43*
    - paragraph **Evaluated data**, *page 45*

      Reports about the dataset presented in [105].

Bibliography

[1] M'hammed Abdous, He Wu, and Cherng-Jyh Yen, *Using data mining for predicting relationships between online question theme and final grade*, Journal of Educational Technology & Society **15** (2012), no. 3, 77.

[2] Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo, *Kanban in software development: A systematic literature review*, 2013 39th Euromicro conference on software engineering and advanced applications, IEEE, 2013, pp. 9–16.

[3] Ma'en Aljezawi and Mohammed Albashtawy, *Quiz game teaching format versus didactic lectures*, British Journal of Nursing **24** (2015), no. 2, 86–92.

[4] STEM Allicance, *Introduction to stem education*, online publication, available at `http://www.stemalliance.eu/documents/99712/104016/STEM-Alliance-Fact-Sheet/4ae068f4-ca07-459a-92c9-17ff305341b1`, retrieved 08.2019.

[5] Mario Amelung, Michael Piotrowski, and Dietmar Rösner, *Educomponents: Experiences in e-assessment in computer science education*, Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education, 2006, pp. 88–92.

[6] Anna Maria Angelone and Pierpaolo Vittorini, *The automated grading of r code snippets: Preliminary results in a course of health informatics*, International Conference in Methodologies and intelligent Systems for Techhnology Enhanced Learning, Springer, 2019, pp. 19–27.

[7] Glenda Anthony, *Active learning in a constructivist framework*, Educational studies in mathematics **31** (1996), no. 4, 349–369.

[8] Kimberly E Arnold and Matthew D Pistilli, *Course signals at purdue: Using learning analytics to increase student success*, Proceedings of the 2nd international conference on learning analytics and knowledge, ACM, 2012, pp. 267–270.

[9] Elliot Aronson et al., *The jigsaw classroom.*, Sage, 1978.

[10] James Arvanitakis, *Massification and the large lecture theatre: From panic to excitement*, Higher Education **67** (2014), no. 6, 735–745.

[11] John T Avella, Mansureh Kebritchi, Sandra G Nunn, and Therese Kanai, *Learning analytics methods, benefits, and challenges in higher education: A systematic literature review.*, Online Learning **20** (2016), no. 2, 13–29.

[12] James Bach, *Exploratory testing explained*, 2003.

[13] Paul Baepler, JD Walker, and Michelle Driessen, *It's not about seat time: Blending, flipping, and efficiency in active learning classrooms*, Computers & Education **78** (2014), 227–236.

[14] Albert Bandura, *Self-efficacy mechanism in human agency.*, American psychologist **37** (1982), no. 2, 122.

[15] Mustafa Baser, *Promoting conceptual change through active learning using open source software for physics simulations*, Australasian Journal of Educational Technology **22** (2006), no. 3, 336–354.

[16] Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss, *A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains*, The annals of mathematical statistics **41** (1970), no. 1, 164–171.

[17] Kent Beck, *Embracing change with extreme programming*, Computer **32** (1999), no. 10, 70–77.

[18] Kelly Bedard and Peter Kuhn, *Where class size really matters: Class size and student ratings of instructor effectiveness*, Economics of Education Review **27** (2008), no. 3, 253–265.

[19] Hefer Bembenutty and Barry J Zimmerman, *The relation of motivational beliefs and self-regulatory processes to homework completion and academic achievement.*, Proceedings of the Annual Meeting American Educational Research Association (Chicago, IL, April 21-25, 2003), American Educational Research Association (AERA), 2003.

[20] Kristen L Benè and George Bergus, *When learners become teachers*, Family medicine **46** (2014), 783–787.

[21] Lucas Bester, Gregg Muller, Brendon Munge, Marcus Morse, and Noel Meyers, *Those who teach learn: Near-peer teaching as outdoor environmental education curriculum and pedagogy*, Journal of Outdoor and Environmental Education **20** (2017), no. 1, 35–46.

[22] John B Biggs, *Teaching for quality learning at university: What the student does*, McGraw-hill education (UK), 2011.

[23] Susan Blackley and Jennifer Howell, *A stem narrative: 15 years in the making.*, Australian Journal of Teacher Education **40** (2015), no. 7, 8.

[24] John Boli, Francisco O Ramirez, and John W Meyer, *Explaining the origins and expansion of mass education*, Comparative education review **29** (1985), no. 2, 145–170.

[25] Raffaella Borasi, *Exploring mathematics through the analysis of errors*, For the learning of Mathematics **7** (1987), no. 3, 2–8.

[26] Andreas Born, *Predicting students' assignment performance to personalize blended learning*, master thesis, Institute of Computer Science, LMU, Munich, 2017.

[27] Denny Borsboom, Gideon J Mellenbergh, and Jaap Van Heerden, *The theoretical status of latent variables.*, Psychological review **110** (2003), no. 2, 203.

[28] Patrick Borunda, Chris Brewer, and Cesim Erten, *Gspim: Graphical visualization tool for mips assembly programming and simulation*, ACM SIGCSE Bulletin **38** (2006), no. 1, 244–248.

[29] Matt Bower and Panos Vlachopoulos, *A critical analysis of technology-enhanced learning design frameworks*, British Journal of Educational Technology **49** (2018), no. 6, 981–997.

[30] David Shane Brewer and Kurt Becker, *Online homework effectiveness for underprepared and repeating college algebra students*, Journal of Computers in Mathematics and Science Teaching **29** (2010), no. 4, 353–371.

[31] Jim Broadbent and Walter L Poon, *Self-regulated learning strategies & academic achievement in online higher education learning environments: A systematic review*, The Internet and Higher Education **27** (2015), 1–13.

[32] Peter Brusilovsky, Eduardo Calabrese, Jozef Hvorecky, Anatoly Kouchnirenko, and Philip Miller, *Mini-languages: A way to learn programming principles*, Education and information technologies **2** (1997), no. 1, 65–83.

[33] Samuel P Bryfczynski, Rebecca Brown, Josiah Hester, Andrew Herrmann, Danielle L Koch, Melanie M Cooper, and Nathaniel P Grove, *urespond: ipad as interactive, personal response system*, Journal of Chemical Education **91** (2014), no. 3, 357–363.

[34] Kimberly Jordan Burch and Yu-Ju Kuo, *Traditional vs. online homework in college algebra*, Mathematics and computer education **44** (2010), no. 1, 53–63.

[35] Noel Burch, *The four stages for learning any new skill*, Gordon Training International, CA, 1970.

[36] Jennifer A Butler, *Use of teaching methods within the lecture format*, Medical teacher **14** (1992), no. 1, 11–25.

[37] Wagner Cambruzzi, Sandro José Rigo, and Jorge L V Barbosa, *Dropout prediction and reduction in distance education courses with the learning analytics multitrail approach.*, J. UCS **21** (2015), no. 1, 23–47.

[38] David Carless, *Differing perceptions in the feedback process*, Studies in higher education **31** (2006), no. 2, 219–233.

[39] John M Carroll, Robert L Mack, Clayton H Lewis, Nancy L Grischkowsky, and Scott R Robertson, *Exploring exploring a word processor*, Human-computer interaction **1** (1985), no. 3, 283–307.

[40] Adam S Carter, Christopher D Hundhausen, and Olusola Adesope, *The normalized programming state model: Predicting student performance in computing courses based on programming behavior*, Proceedings of the eleventh annual International Conference on International Computing Education Research, ACM, 2015, pp. 141–150.

[41] Mohamed Amine Chatti, Anna Lea Dyckhoff, Ulrik Schroeder, and Hendrik Thüs, *A reference model for learning analytics*, International Journal of Technology Enhanced Learning **4** (2013), no. 5-6, 318–331.

[42] Kwangsu Cho and Charles MacArthur, *Learning by reviewing.*, Journal of Educational Psychology **103** (2011), no. 1, 73.

[43] Samuel PM Choi, Sze Sing Lam, Kam Cheong Li, and Billy TM Wong, *Learning analytics at low cost: At-risk student prediction with clicker data and systematic proactive interventions*, Journal of Educational Technology & Society **21** (2018), no. 2, 273–290.

[44] Joseph Chow, Ada Tse, and Christine Armatas, *Comparing trained and untrained teachers on their use of lms tools using the rasch analysis*, Computers & Education **123** (2018), 124–137.

[45] Jere Confrey, *Chapter 1: A review of the research on student conceptions in mathematics, science, and programming*, Review of research in education **16** (1990), no. 1, 3–56.

[46] Harris Cooper, *Synthesis of research on homework*, Educational leadership **47** (1989), no. 3, 85–91.

[47] Harris Cooper and Jeffrey C Valentine, *Using research to answer practical questions about homework*, Educational psychologist **36** (2001), no. 3, 143–153.

[48] James L Cooper and Pamela Robinson, *The argument for making large classes seem small*, New directions for teaching and learning **81** (2000), no. 81, 5–16.

[49] Linda Corrin and Paula de Barba, *Exploring students' interpretation of feedback delivered through learning analytics dashboards*, Proceedings of the ascilite conference, 2014, pp. 629–633.

[50] Ulrike Cress, *Mass collaboration-an emerging field for cscl research*, To See the World and a Grain of Sand: Learning across Levels of Space, Time, and Scale: CSCL 2013 Conference Proceedings Volume 1 — Full Papers & Symposia, International Society of the Learning Sciences, 2013, pp. 557–563.

[51] Ulrike Cress and Gerhard Fischer, *Mass collaboration with social software in tel*, Technology Enhanced Learning, Springer, 2017, pp. 59–67.

[52] Al Cripps, *Using artificial neural nets to predict academic performance*, Proceedings of the 1996 ACM Symposium on Applied Computing, ACM, 1996, pp. 33–37.

[53] David Crosier, Peter Birch, Olga Davydovskaia, Daniela Kocanova, and Teodora Parveva, *Modernisation of higher education in europe: Academic staff–2017. eurydice report.*, Education, Audiovisual and Culture Executive Agency, European Commission (2017), 1–172.

[54] Julie Crough and Christopher Love, *Improving student engagement and self-regulated learning through technology-enhanced student partnerships*, Proceedings of the International Conference on Information, Communication Technologies in Education, 2019.

[55] Nada Dabbagh and Anastasia Kitsantas, *Personal learning environments, social media, and self-regulated learning: A natural formula for connecting formal and informal learning*, The Internet and higher education **15** (2012), no. 1, 3–8.

[56] Eden Dahlstrom, D Christopher Brooks, and Jacqueline Bichsel, *The current ecosystem of learning management systems in higher education: Student, faculty, and it perspectives*, 2014.

[57] Mette Trier Damgaard and Helena Skyt Nielsen, *Nudging in education*, Economics of Education Review **64** (2018), 313 – 342.

[58] Maddalena Davoli and Horst Entorf, *The pisa shock, socioeconomic inequality, and school reforms in germany*, Tech. report, IZA Policy Paper, 2018.

[59] Manuela Delfino and Donatella Persico, *Unfolding the potential of ict for srl development*, Self-Regulated Learning in Technology Enhanced Learning Environments, Brill Sense, 2011, pp. 51–74.

[60] Önder Demir, Aykut Soysal, Ahmet Arslan, Burcu Yürekli, and Özgür Yılmazel, *Automatic grading system for programming homework*, Computer Science Education: Innovation and Technology, CSEIT (2010).

[61] Neset Demirci, *University students' perceptions of web-based vs. paper-based homework in a general physics course.*, Online Submission **3** (2007), no. 1, 29–34.

[62] Vanessa P Dennen and Kerry J Burner, *The cognitive apprenticeship model in educational practice*, Handbook of research on educational communications and technology **3** (2008), 425–439.

[63] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx, *Understanding the syntax barrier for novices*, Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, ACM, 2011, pp. 208–212.

[64] Pierre Dillenbourg, *Integrating technologies into educational ecosystems*, Distance Education **29** (2008), no. 2, 127–140.

[65] Pierre Dillenbourg and Fabrice Hong, *The mechanics of cscl macro scripts*, International Journal of Computer-Supported Collaborative Learning **3** (2008), no. 1, 5–23.

[66] Pierre Dillenbourg and Patrick Jermann, *Designing integrative scripts*, Scripting computer-supported collaborative learning, Springer, 2007, pp. 275–301.

[67] ———, *Technology for classroom orchestration*, New science of learning, Springer, 2010, pp. 525–552.

[68] Pierre Dillenbourg and Pierre Tchounikine, *Flexibility in macro-scripts for computer-supported collaborative learning*, Journal of computer assisted learning **23** (2007), no. 1, 1–13.

[69] Filip Dochy, Mien Segers, and Dominique Sluijsmans, *The use of self-, peer and co-assessment in higher education: A review*, Studies in Higher education **24** (1999), no. 3, 331–350.

[70] Steven Dostert, *Predicting the learning behaviour of students from their weekly work assignments*, master thesis, Institute of Computer Science, LMU, Munich, 2017.

[71] Itiel E Dror, *Technology enhanced learning: The good, the bad, and the ugly*, Pragmatics & Cognition **16** (2008), no. 2, 215–223.

[72] Maralyn Druce and Stella Howden, *New perspectives on health professions students'e-learning: Looking through the lens of the "visitor and resident" model*, Medical teacher **39** (2017), no. 7, 704–709.

[73] Ed Dubinsky, *Meaning and formalism in mathematics*, International Journal of Computers for Mathematical Learning **5** (2000), no. 3, 211–240.

[74] Christof Ebert and Maria Paasivaara, *Scaling agile*, IEEE Software **34** (2017), no. 6, 98–103.

[75] Alex Edgcomb, Frank Vahid, Roman Lysecky, and Susan Lysecky, *Getting students to earnestly do reading, studying, and homework in an introductory programming class*, Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, ACM, 2017, pp. 171–176.

[76] Laurie D Edwards, *Microworlds as representations*, Computers and exploratory learning, Springer, 1995, pp. 127–154.

[77] William H Edwards, *Motor learning and control: From theory to practice*, Cengage Learning, 2010.

[78] Asmaa Elbadrawy, R Scott Studham, and George Karypis, *Collaborative multi-regression models for predicting students' performance in course activities*, Proceedings of the Fifth International Conference on Learning Analytics And Knowledge, ACM, 2015, pp. 103–107.

[79] Julian G Elliott, Wilma CM Resing, and Jens F Beckmann, *Dynamic assessment: A case of unfulfilled potential?*, Educational Review **70** (2018), no. 1, 7–17.

[80] Carol Evans, *Making sense of assessment feedback in higher education*, Review of educational research **83** (2013), no. 1, 70–120.

[81] Daniel J Exeter, Shanthi Ameratunga, Matiu Ratima, Susan Morton, Martin Dickson, Dennis Hsu, and Rod Jackson, *Student engagement in very large classes: The teachers' perspective*, Studies in Higher Education **35** (2010), no. 7, 761–775.

[82] Statistics Explained, *Tertiary education statistics*, online publication, available at `https://ec.europa.eu/eurostat/statistics-explained/index.php/Tertiary_education_statistics`, retrieved 02.09.2019.

[83] Nancy Falchikov and Judy Goldfinch, *Student peer assessment in higher education: A meta-analysis comparing peer and teacher marks*, Review of educational research **70** (2000), no. 3, 287–322.

[84] Stefano Federici, *A minimal, extensible, drag-and-drop implementation of the c programming language*, Proceedings of the 2011 conference on Information technology education, 2011, pp. 191–196.

[85] Rebecca Ferguson, Andrew Brasher, Doug Clow, Adam Cooper, Garron Hillaire, Jenna Mittelmeier, Bart Rienties, Thomas Ullmann, and Riina Vuorikari, *Research evidence on the use of learning analytics: Implications for education policy*, Joint Research Centre (2016).

[86] Rebecca Ferguson and Doug Clow, *Where is the evidence? a call to action for learning analytics*, Proceedings of the seventh international learning analytics & knowledge conference, 2017, pp. 56–65.

[87] Martin Fowler and Matthew Foemmel, *Continuous integration*, 2006.

[88] Martin Fowler, Jim Highsmith, et al., *The agile manifesto*, Software Development **9** (2001), no. 8, 28–35.

[89] Joseph S Francisco, Gayle Nicoll, and Marcella Trautmann, *Integrating multiple teaching methods into a general chemistry classroom*, Journal of chemical education **75** (1998), no. 2, 210.

[90] Scott Freeman, Sarah L Eddy, Miles McDonough, Michelle K Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth, *Active learning increases student performance in science, engineering, and mathematics*, Proceedings of the National Academy of Sciences **111** (2014), no. 23, 8410–8415.

[91] Herb Fynewever, *A comparison of the effectiveness of web-based and paper-based homework for general chemistry*, The Chemical Educator **13** (2008), no. 4, 264–269.

[92] Dragan Gašević, Shane Dawson, and George Siemens, *Let's not forget: Learning analytics are about learning*, TechTrends **59** (2015), no. 1, 64–71.

[93] Gaetano Geck, Artur Ljulin, Sebastian Peter, Jonas Schmidt, Fabian Vehlken, and Thomas Zeume, *Introduction to iltis: An interactive, web-based system for teaching logic*, Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, 2018, pp. 141–146.

[94] John Gerard Scott Goldie, *Connectivism: A knowledge learning theory for the digital age?*, Medical teacher **38** (2016), no. 10, 1064–1069.

[95] Barbara Goldschmid and Marcel L Goldschmid, *Peer teaching in higher education: A review*, Higher education **5** (1976), no. 1, 9–33.

[96] Anabela Gomes and António José Mendes, *Learning to program-difficulties and solutions*, International Conference on Engineering Education–ICEE, vol. 2007, 2007.

[97] Anatoliy Gruzd, Caroline Haythornthwaite, Drew Paulin, Sarah Gilbert, and Marc Esteve Del Valle, *Uses and gratifications factors for social media use in teaching: Instructors' perspectives*, New Media & Society **20** (2018), no. 2, 475–494.

[98] Huseyin Guruler, Ayhan Istanbullu, and Mehmet Karahasan, *A new student performance analysing system using knowledge discovery in higher educational databases*, Computers & Education **55** (2010), no. 1, 247–254.

[99] Allyson F Hadwin, John C Nesbit, Dianne Jamieson-Noel, Jillianne Code, and Philip H Winne, *Examining trace data to explore self-regulated learning*, Metacognition and Learning **2** (2007), no. 2-3, 107–124.

[100] Linda Harasim, *Learning theory and online technologies*, Routledge, 2017.

[101] John Hattie, *Visible learning: A synthesis of over 800 meta-analyses relating to achievement*, routledge, 2008.

[102] John Hattie and Helen Timperley, *The power of feedback*, Review of educational research **77** (2007), no. 1, 81–112.

[103] Niels Heller and François Bry, *Predicting learners' behaviours to get it wrong*, International Conference in Methodologies and intelligent Systems for Techhnology Enhanced Learning, Springer, 2018, pp. 12–19.

[104] ———, *Collaborative correction in mass education as a social media application*, Proceedings of the 6th European Conference on Social Media (ECSM 2019), ACPI, 2019, pp. 102 – 110.

[105] ———, *Learning by fiddling: Patterns of behaviour in formal language learning*, International Conference in Methodologies and intelligent Systems for Techhnology Enhanced Learning, Springer, 2019, pp. 28–36.

[106] ———, *Nudging by predicting: A case study*, Proceedings of the 11th International Conference on Computer Supported Education - Volume 2: CSEDU, SciTePress, 2019, pp. 236–243.

[107] ———, *Organizing peer correction in tertiary stem education: An approach and its evaluation*, International Journal of Engineering Pedagogy (iJEP) **9** (2019), no. 4, 16–32.

[108] ———, *Peer teaching in tertiary stem education: A case study*, The Challenges of the Digital Transformation in Education - Proceedings of the 21st International Conference on Interactive Collaborative Learning (ICL2018), vol. 2, Springer, 25-28 September 2018, pp. 87–98.

[109] Niels Heller, Sebastian Mader, and François Bry, *Backstage: A versatile platform supporting learning and teaching format composition*, Proceedings of the 18th Koli Calling International Conference on Computing Education Research, ACM, 2018.

[110] Niels Heller, Sebastian Mader, and François Bry, *More than the sum of its parts: Designing learning formats from core components*, Proceedings of the 34th ACM/SIGAPP Symposium On Applied Computing, ACM, 2019, pp. 1 – 2.

[111] Michael Henderson, Neil Selwyn, and Rachel Aston, *What works and why? student perceptions of 'useful'digital technology in university teaching and learning*, Studies in Higher Education **42** (2017), no. 8, 1567–1579.

[112] Maxim Hendriks, Cezary Kaliszyk, Femke Van Raamsdonk, and Freek Wiedijk, *Teaching logic using a state-of-the-art proof assistant.*, Acta Didactica Napocensia **3** (2010), no. 2, 35–48.

[113] Robert Holmgren and Sigurd Johansson, *Reducing dropouts in online education-group tutoring in virtual seminars and support structures.*, Online Submission (2012), 971–978.

[114] David J Hornsby and Ruksana Osman, *Massification in higher education: Large classes and student learning*, Higher education **67** (2014), no. 6, 711–719.

[115] Paul Horwitz and Bill Barowy, *Designing and using open-ended software to promote conceptual change*, Journal of Science Education and Technology **3** (1994), no. 3, 161–185.

[116] Ying-Shao Hsu, Hsin-Kai Wu, and Fu-Kwun Hwang, *Fostering high school students' conceptual understandings about seasons: The design of a technology-enhanced learning environment*, Research in Science Education **38** (2008), no. 2, 127–147.

[117] M Qamarul Islam and Moti L Tiku, *Multiple linear regression model under nonnormality*, Communications in Statistics-Theory and Methods **33** (2005), no. 10, 2443–2467.

[118] ISO, *Pascal*, Standard, International Organization for Standardization, Geneva, CH, 1990.

[119] David Janzen and Hossein Saiedian, *Test-driven development concepts, taxonomy, and future direction*, Computer **38** (2005), no. 9, 43–50.

[120] Sanna Järvelä and Allyson Hadwin, *Promoting and researching adaptive regulation: New frontiers for cscl research*, 2015.

[121] Craig William Jenkins, *Microworlds: Building powerful ideas in the secondary school.*, Online Submission, US-China Education Review A 9 (2012), 796–803.

[122] Tony Jenkins, *On the difficulty of learning to program*, Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, vol. 4, Citeseer, 2002, pp. 53–58.

[123] Reynol Junco, *The relationship between frequency of facebook use, participation in facebook activities, and student engagement*, Computers & education **58** (2012), no. 1, 162–171.

[124] Veselin Jungic, Deborah Kent, and Petra Menz, *On online assignments in a calculus class*, Journal of University Teaching & Learning Practice **9** (2012), no. 1, 3.

[125] Daniel Kahneman, Jack L Knetsch, and Richard H Thaler, *Experimental tests of the endowment effect and the coase theorem*, Journal of political Economy **98** (1990), no. 6, 1325–1348.

[126] Cody Kalina and KC Powell, *Cognitive and social constructivism: Developing tools for an effective classroom*, Education **130** (2009), no. 2, 241–250.

[127] Jonathan Kaplan, *Co-regulation in technology enhanced learning environments*, International Workshop on Learning Technology for Education in Cloud, Springer, 2014, pp. 72–81.

[128] Mümine Kaya and Selma Ayşe Özel, *Integrating an online compiler and a plagiarism detection tool into the moodle distance education system for easy assessment of programming*

assignments, Computer Applications in Engineering Education **23** (2015), no. 3, 363–373.

[129] Mansureh Kebritchi, Angie Lipschuetz, and Lilia Santiague, *Issues and challenges for teaching successful online courses in higher education: A literature review*, Journal of Educational Technology Systems **46** (2017), no. 1, 4–29.

[130] Galina Keil, *Generierung didaktischer fehlermeldungen für minisprachen, institute of computer science, lmu, munich*, Bachelorarbeit/bachelor thesis, 2019.

[131] Alice Kerly, Richard Ellis, and Susan Bull, *Calmsystem: a conversational agent for learner modelling*, International Conference on Innovative Techniques and Applications of Artificial Intelligence, Springer, 2007, pp. 89–102.

[132] Hanan Khalil and Martin Ebner, *Moocs completion rates and possible methods to improve retention-a literature review*, EdMedia+ Innovate Learning, Association for the Advancement of Computing in Education (AACE), 2014, pp. 1305–1313.

[133] Adrian Kirkwood and Linda Price, *Technology-enhanced learning and teaching in higher education: what is 'enhanced'and how do we know? a critical literature review*, Learning, media and technology **39** (2014), no. 1, 6–36.

[134] David Kirshner, *The visual syntax of algebra*, Journal for Research in Mathematics Education (1989), 274–287.

[135] Anastasia Kitsantas and Barry J Zimmerman, *College students' homework and academic achievement: The mediating role of self-regulatory beliefs*, Metacognition and Learning **4** (2009), no. 2, 97–110.

[136] René F Kizilcec, Chris Piech, and Emily Schneider, *Deconstructing disengagement: Analyzing learner subpopulations in massive open online courses*, Proceedings of the third international conference on learning analytics and knowledge, ACM, 2013, pp. 170–179.

[137] Lars Kobbe, Armin Weinberger, Pierre Dillenbourg, Andreas Harrer, Raija Hämäläinen, Päivi Häkkinen, and Frank Fischer, *Specifying computer-supported collaboration scripts*, International Journal of Computer-Supported Collaborative Learning **2** (2007), no. 2-3, 211–224.

[138] Alfie Kohn, *The case against grades*, Educational Leadership **69** (2011), no. 3, 28–33.

[139] Edward C Kokkelenberg, Michael Dillon, and Sean M Christy, *The effects of class size on student grades at a public university*, Economics of Education Review **27** (2008), no. 2, 221–233.

[140] Justin Kruger and David Dunning, *Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments.*, Journal of personality and social psychology **77** (1999), no. 6, 1121.

[141] Anna Kruse and Rob Pongsajapan, *Student-centered learning analytics*, CNDLS Thought Papers (2012), 1–9.

[142] Daniel G Krutka and Jeffrey P Carpenter, *Why social media must have a place in schools*, Kappa Delta Pi Record **52** (2016), no. 1, 6–10.

[143] J Richard Landis and Gary G Koch, *The measurement of observer agreement for categorical data*, biometrics (1977), 159–174.

[144] Celeste Lawson, Colin Beer, Dolene Rossi, Teresa Moore, and Julie Fleming, *Identification of 'at risk'students using learning analytics: The ethical dilemmas of intervention*

strategies in a higher education institution, Educational Technology Research and Development **64** (2016), no. 5, 957–968.

[145] Mark JW Lee and Catherine McLoughlin, *Teaching and learning in the web 2.0 era: Empowering students through learner-generated content*, International journal of instructional technology and distance learning **4** (2007), no. 10, 1–17.

[146] Elisabeth Lempa, *Coconut 2: Concurrently virtualising user code compilation*, Bachelorarbeit/bachelor thesis, 2018.

[147] Lisa-Angelique Lim, Sheridan Gentili, Abelardo Pardo, Vitomir Kovanović, Alexander Whitelock-Wainwright, Dragan Gašević, and Shane Dawson, *What changes, and for whom? a study of the impact of learning analytics-based process feedback in a large course*, Learning and Instruction (2019), 101202.

[148] Raymond Lister, Elizabeth S Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, et al., *A multi-national study of reading and tracing skills in novice programmers*, ACM SIGCSE Bulletin, vol. 36, ACM, 2004, pp. 119–150.

[149] Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller, *Turkit: Human computation algorithms on mechanical turk*, Proceedings of the 23nd annual ACM symposium on User interface software and technology, ACM, 2010, pp. 57–66.

[150] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister, *Relationships between reading, tracing and writing skills in introductory programming*, Proceedings of the fourth international workshop on computing education research, ACM, 2008, pp. 101–112.

[151] Betty Love, Angie Hodge, Neal Grandgenett, and Andrew W Swift, *Student learning and perceptions in a flipped linear algebra course*, International Journal of Mathematical Education in Science and Technology **45** (2014), no. 3, 317–324.

[152] Joost Lowyck, *Bridging learning theories and technology-enhanced environments: A critical appraisal of its history*, Handbook of research on educational communications and technology, Springer, 2014, pp. 3–20.

[153] Kristi Lundstrom and Wendy Baker, *To give is better than to receive: The benefits of peer review to the reviewer's own writing*, Journal of second language writing **18** (2009), no. 1, 30–43.

[154] Richard Lynch and Myron Dembo, *The relationship between self-regulation and online learning in a blended learning context*, The International Review of Research in Open and Distributed Learning **5** (2004), no. 2.

[155] Sebastian Mader, *Technology for interactivity and engagement in large class education*, Dissertation, 2020, to appear.

[156] Sebastian Mader, Niels Heller, and François Bry, *Adding narrative to gamification and educational games with generic templates*, Proceedings of the 18th European Conference on e-Learning (ECEL 2019), ACPI, 2019, pp. 360–368.

[157] PD Magnus, Tim Button, J Robert Loftis, Aaron Thomas-Bolduc, and Richard Zach, *forall x: Calgary remix*.

[158] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond, *The scratch programming language and environment*, ACM Transactions on Computing Education (TOCE) **10** (2010), no. 4, 16.

[159] Thom Markham, *Project based learning a bridge just far enough*, Teacher librarian **39** (2011), no. 2, 38.

[160] Tatiana Markova, Irina Glazkova, and Elena Zaborova, *Quality issues of online distance learning*, Procedia-Social and Behavioral Sciences **237** (2017), 685–691.

[161] Geoff N Masters, *A rasch model for partial credit scoring*, Psychometrika **47** (1982), no. 2, 149–174.

[162] Hunter A McAllister, *Self-serving bias in the classroom: Who shows it? who knows it?*, Journal of Educational Psychology **88** (1996), no. 1, 123.

[163] Barbara Means, Yuki Toyama, Robert Murphy, Marianne Bakia, and Karla Jones, *Evaluation of evidence-based practices in online learning: A meta-analysis and review of online learning studies*, (2009).

[164] Agathe Merceron and Kalina Yacef, *Interestingness measures for association rules in educational data*, Educational Data Mining 2008, 2008.

[165] Mariel Miller and Allyson Hadwin, *Scripting and awareness tools for regulating collaborative learning: Changing the landscape of support in cscl*, Computers in Human Behavior **52** (2015), 573–588.

[166] James Monks and Robert M Schmidt, *The impact of class size on outcomes in higher education*, The BE Journal of Economic Analysis & Policy **11** (2011), no. 1.

[167] Tamara J Moore and Karl A Smith, *Advancing the state of the art of stem integration*, Journal of STEM Education: Innovations and Research **15** (2014), no. 1, 5.

[168] Nitsa Movshovitz-Hadar, Orit Zaslavsky, and Shlomo Inbar, *An empirical classification model for errors in high school mathematics*, Journal for research in mathematics Education (1987), 3–14.

[169] Catherine Mulryan-Kyne, *Teaching large classes at college and university level: Challenges and opportunities*, Teaching in Higher Education **15** (2010), no. 2, 175–185.

[170] Mary A Newman, *An analysis of sixth-grade pupil's error on written mathematical tasks*, Victorian Institute for Educational Research Bulletin **39** (1977), 31–43.

[171] David Nicol, *From monologue to dialogue: Improving written feedback processes in mass higher education*, Assessment & Evaluation in Higher Education **35** (2010), no. 5, 501–517.

[172] David Nicol, Avril Thomson, and Caroline Breslin, *Rethinking feedback practices in higher education: A peer review perspective*, Assessment & Evaluation in Higher Education **39** (2014), no. 1, 102–122.

[173] Melanie Njoo and Ton De Jong, *Exploratory learning with a computer simulation for control theory: Learning processes and instructional support*, Journal of research in science teaching **30** (1993), no. 8, 821–844.

[174] Jonathan A Obar and Steven S Wildman, *Social media definition and the governance challenge-an introduction to the special issue*, Obar, JA and Wildman, S.(2015). Social media definition and the governance challenge: An introduction to the special issue. Telecommunications policy **39** (2015), no. 9, 745–750.

[175] VO Oladokun, AT Adebanjo, and OE Charles-Owaba, *Predicting students' academic performance using artificial neural network: A case study of an engineering course*, The Pacific Journal of Science and Technology **9** (2008), no. 1, 72–79.

[176] Daniel FO Onah, Jane Sinclair, and Russell Boyatt, *Dropout rates of massive open online courses: Behavioural patterns*, EDULEARN14 proceedings **1** (2014), 5825–5834.

[177] John D Ophus and Jason T Abbitt, *Exploring the potential perceptions of social networking systems in university courses*, Journal of Online Learning and Teaching **5** (2009), no. 4, 639–648.

[178] P Orsmond and S Merry, *Processing tutor feedback: A consideration of qualitative differences in learning outcomes for high and non-high achieving students*, Fostering Communities of Learners, 13th EARLI conference, August, 2009, pp. 25–29.

[179] Michael Osborne, *Increasing or widening participation in higher education?: A european overview*, European journal of education **38** (2003), no. 1, 5–24.

[180] Susan W Palocsay and Scott P Stevens, *A study of the effectiveness of web-based homework in teaching undergraduate business statistics*, Decision Sciences Journal of Innovative Education **6** (2008), no. 2, 213–232.

[181] Zacharoula Papamitsiou and Anastasios A Economides, *Learning analytics and educational data mining in practice: A systematic literature review of empirical evidence*, Journal of Educational Technology & Society **17** (2014), no. 4, 49–64.

[182] Seymour Papert, *Microworlds: Transforming education*, Artificial intelligence and education, vol. 1, Ablex Utrecht, 1987, pp. 79–94.

[183] Abelardo Pardo, Kathryn Bartimote, Simon Buckingham Shum, Shane Dawson, Jing Gao, Dragan Gašević, Steve Leichtweis, Danny Liu, Roberto Martínez-Maldonado, Negin Mirriahi, et al., *Ontask: Delivering data-informed, personalized learning support actions*, Journal of Learning Analytics **5** (2018), no. 3, 235–249.

[184] Zachary A Pardos, Ryan SJD Baker, Maria OCZ San Pedro, Sujith M Gowda, and Supreeth M Gowda, *Affective states and state tests: Investigating how affect throughout the school year predicts end of year learning outcomes*, Proceedings of the Third International Conference on Learning Analytics and Knowledge, ACM, 2013, pp. 117–124.

[185] Yeonjeong Park and I-H Jo, *Development of the learning analytics dashboard to support students' learning performance*, Journal of Universal Computer Science **21** (2015), no. 1, 110.

[186] Laurie L Parker and G Marc Loudon, *Case study using online homework in undergraduate organic chemistry: Results and student attitudes*, Journal of Chemical Education **90** (2012), no. 1, 37–44.

[187] Andrea M Pascarella, *The influence of web-based homework on quantitative problem-solving in a university physics class*, Proc. NARST Annual Meeting, vol. 4, 2004, pp. 19–28.

[188] Lindsay Paterson, *Higher education and european regionalism*, Pedagogy Culture and Society **9** (2001), no. 2, 133–160.

[189] Paul R Pintrich, *The role of goal orientation in self-regulated learning*, Handbook of self-regulation, Elsevier, 2000, pp. 451–502.

[190] Paul R Pintrich and Elisabeth V De Groot, *Motivational and self-regulated learning components of classroom academic performance.*, Journal of educational psychology **82** (1990), no. 1, 33.

[191] Nea Pirttinen, Vilma Kangas, Henrik Nygren, Juho Leinonen, and Arto Hellas, *Analysis of students' peer reviews to crowdsourced programming assignments*, Proceedings of the

18th Koli Calling International Conference on Computing Education Research, ACM, 2018, p. 21.

[192] George J Posner, Kenneth A Strike, Peter W Hewson, and William A Gertzog, *Accommodation of a scientific conception: Toward a theory of conceptual change*, Science education **66** (1982), no. 2, 211–227.

[193] Robert Pospisil, *Lerngruppenbildung anhand von kompetenzschätzung mittels bayescher netze*, master thesis, Institute of Computer Science, LMU, Munich, 2019.

[194] Edward Price, Fred Goldberg, Steve Robinson, and Michael McKean, *Validity of peer grading using calibrated peer review in a guided-inquiry, conceptual physics course*, Physical Review Physics Education Research **12** (2016), no. 2, 020145.

[195] Margaret Price, Karen Handley, Jill Millar, and Berry O'donovan, *Feedback: All that effort, but what is the effect?*, Assessment & Evaluation in Higher Education **35** (2010), no. 3, 277–289.

[196] Michael Prince, *Does active learning work? a review of the research*, Journal of engineering education **93** (2004), no. 3, 223–231.

[197] David Pritchard and Troy Vasiga, *Cs circles: An in-browser python course for beginners*, Proceeding of the 44th ACM technical symposium on Computer science education, 2013, pp. 591–596.

[198] Yizhou Qian and James Lehman, *Students' misconceptions and other difficulties in introductory programming: A literature review*, ACM Transactions on Computing Education (TOCE) **18** (2017), no. 1, 1.

[199] Lawrence R Rabiner, *A tutorial on hidden markov models and selected applications in speech recognition*, Proceedings of the IEEE **77** (1989), no. 2, 257–286.

[200] Hendrik Radatz, *Error analysis in mathematics education*, Journal for Research in mathematics Education (1979), 163–172.

[201] Christopher R Rakes and Robert N Ronau, *Rethinking mathematics misconceptions: Using knowledge structures to explain systematic errors within and across content domains*, International Journal of Research in Education and Science **5** (2018), no. 1, 1–21.

[202] Patient Rambe, *Critical discourse analysis of collaborative engagement in facebook postings*, Australasian Journal of Educational Technology **28** (2012), no. 2.

[203] Darshanand Ramdass and Barry J Zimmerman, *Effects of self-correction strategy training on middle school students' self-efficacy, self-evaluation, and mathematics division learning*, Journal of advanced academics **20** (2008), no. 1, 18–41.

[204] N Sanjay Rebello, Lili Cui, Andrew G Bennett, Dean A Zollman, and Darryl J Ozimek, *Transfer of learning in problem solving in the context of mathematics and physics*, Learning to solve complex scientific problems, vol. 223, Lawrence Erlbaum Associates Mahwah, NJ, 2007.

[205] Sasha A Reese, *Online learning environments in higher education: Connectivism vs. dissociation*, Education and information technologies **20** (2015), no. 3, 579–588.

[206] Alexander Renkl, Heinz Mandl, and Hans Gruber, *Inert knowledge: Analyses and remedies*, Educational Psychologist **31** (1996), no. 2, 115–121.

[207] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silver-

man, et al., *Scratch: Programming for all*, Communications of the ACM **52** (2009), no. 11, 60–67.

[208] Michelle Richards-Babb, Janice Drelick, Zachary Henry, and Jennifer Robertson-Honecker, *Online homework, help or hindrance? what students think and how they perform.*, Journal of College Science Teaching **40** (2011), no. 4, 70–82.

[209] Lloyd P Rieber, *Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games*, Educational technology research and development **44** (1996), no. 2, 43–58.

[210] John Rieman, *A field study of exploratory learning strategies*, ACM Transactions on Computer-Human Interaction (TOCHI) **3** (1996), no. 3, 189–218.

[211] Anthony Robins, *Learning edge momentum: A new account of outcomes in cs1*, Computer Science Education **20** (2010), no. 1, 37–71.

[212] Susan H Rodger, Anna O Bilska, Kenneth H Leider, Magdalena Procopiuc, Octavian Procopiuc, Jason R Salemme, and Edwin Tsang, *A collection of tools for making automata theory and formal languages come alive*, Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education, 1997, pp. 15–19.

[213] Santiago Rodríguez, José L Pedraza, Antonio G Dopico, Francisco Rosales, and Rafael Méndez, *Computer-based management environment for an assembly language programming laboratory*, Computer Applications in Engineering Education **15** (2007), no. 1, 41–54.

[214] David Romer, *Do students go to class? should they?*, Journal of Economic Perspectives **7** (1993), no. 3, 167–174.

[215] Jeremy Roschelle, Yannis Dimitriadis, and Ulrich Hoppe, *Classroom orchestration: Synthesis*, Computers & Education **69** (2013), 523–526.

[216] Rheta N Rubenstein and Denisse R Thompson, *Learning mathematical symbolism: Challenges and instructional strategies*, The Mathematics Teacher **94** (2001), no. 4, 265.

[217] Ayesha Sadaf, Timothy J Newby, and Peggy A Ertmer, *Exploring pre-service teachers' beliefs about using web 2.0 technologies in k-12 classroom*, Computers & Education **59** (2012), no. 3, 937–945.

[218] Mark E Sanders, *STEM, STEM education, STEMmania*, The Technology Teacher 68(4) (2008), 20–26.

[219] Mirweis Sangin, Gaëlle Molinari, Marc-Antoine Nüssli, and Pierre Dillenbourg, *Facilitating peer knowledge modeling: Effects of a knowledge awareness tool on collaborative learning outcomes and processes*, Computers in Human Behavior **27** (2011), no. 3, 1059–1067.

[220] Sebastian Schaffert, François Bry, Joachim Baumeister, and Malte Kiesel, *Semantic wiki*, Informatik-Spektrum **30** (2007), no. 6, 434–439.

[221] Astrid Schmulian and Stephen Coetzee, *Class absenteeism: Reasons for non-attendance and the effect on academic performance*, Accounting Research Journal **24** (2011), no. 2, 178–194.

[222] Uwe Schöning, *Theoretische informatik*.

[223] Ken Schwaber, *Scrum development process*, Business object design and implementation, Springer, 1997, pp. 117–134.

[224] PH Scott, HM Asoko, and RH Driver, *Teaching for conceptual change: A review of strategies*, Research in physics learning: Theoretical issues and empirical studies (1992), 310–329.

[225] Christopher Seenan, Sivaramkumar Shanmugam, and Jennie Stewart, *Group peer teaching: A strategy for building confidence in communication and teamwork skills in physical therapy students*, Journal of Physical Therapy Education **30** (2016), no. 3, 40–49.

[226] Neil Selwyn, *Minding our language: Why education and technology is full of bullshit… and what might be done about it*, 2016.

[227] John J Shaughnessy, *Long-term retention and the spacing effect in free-recall and frequency judgments*, The American Journal of Psychology (1977), 587–598.

[228] George Siemens (ed.), *1st international conference on learning analytics and knowledge 2011*, 2010.

[229] Katrin Soika and Priit Reiska, *Using concept mapping for assessment in science education*, Journal of Baltic Science Education **13** (2014), no. 5, 662–673.

[230] Cynthia J Solomon and Seymour Papert, *A case study of a young child doing turtle graphics in logo*, Proceedings of the June 7-10, 1976, national computer conference and exposition, 1976, pp. 1049–1056.

[231] Leonard Springer, Mary Elizabeth Stanne, and Samuel S Donovan, *Effects of small-group learning on undergraduates in science, mathematics, engineering, and technology: A meta-analysis*, Review of educational research **69** (1999), no. 1, 21–51.

[232] Gerry Stahl, *Contributions to a theoretical framework for cscl*, Proceedings of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community, 2002, pp. 62–71.

[233] John Stamper, Michael Eagle, Tiffany Barnes, and Marvin Croy, *Experimental evaluation of automatic hint generation for a logic tutor*, International Journal of Artificial Intelligence in Education **22** (2013), no. 1-2, 3–17.

[234] Korbinian Staudacher, *Conception, implementation and evaluation of proof editors for learning*, Bachelor thesis, Institute of Computer Science, LMU, Munich, 2018.

[235] Korbinian Staudacher, Sebastian Mader, and François Bry, *Automated scaffolding and feedback for proof construction: A case study*, Proceedings of the 18th European Conference on e-Learning (ECEL 2019), ACPI, 2019, pp. 542–550.

[236] Binod Sundararajan, *Emergence of the most knowledgeable other (mko): Social network analysis of chat and bulletin board conversations in a cscl system*, Electronic Journal of e-Learning **8** (2010), no. 2, 191–208.

[237] Ruhan Ozkardes Tandogan and Akinoglu Orhan, *The effects of problem-based active learning in science education on students' academic achievement, attitude and concept learning.*, Online Submission **3** (2007), no. 1, 71–81.

[238] Dirk T Tempelaar, Bart Rienties, and Bas Giesbers, *In search for the most informative data for feedback generation: Learning analytics in a data-rich context*, Computers in Human Behavior **47** (2015), 157–167.

[239] RH Thaler and CR Sunstein, *Nudge: improving decisions about health, wealth, and happiness*, New Haven, CT (USA) Yale Univ. Press, 2008.

[240] Christine D Tippett, *Refutation text in science education: A review of two decades of research*, International journal of science and mathematics education **8** (2010), no. 6, 951–970.

[241] Keith Topping, *Peer assessment between students in colleges and universities*, Review of educational Research **68** (1998), no. 3, 249–276.

[242] Ulrich Trautwein and Olaf Köller, *The relationship between homework and achievement—still much of a mystery*, Educational psychology review **15** (2003), no. 2, 115–145.

[243] April R Trees and Michele H Jackson, *The learning environment in clicker classrooms: Student processes of learning and involvement in large university-level courses using student response systems*, Learning, Media and Technology **32** (2007), no. 1, 21–40.

[244] R Untch, *Teaching programming using the karel the robot paradigm realized with a conventional language*, SPONS AGENCY REPORT NO PUB DATE (1990), 225.

[245] Wouter Van Joolingen, *Cognitive tools for discovery learning*, International Journal of Artificial Intelligence in Education (IJAIED) (1998), 385–397.

[246] Stijn Van Laer and Jan Elen, *In search of attributes that support self-regulation in blended learning environments*, Education and Information Technologies **22** (2017), no. 4, 1395–1454.

[247] Silke Vanslambrouck, Chang Zhu, Bram Pynoo, Koen Lombaerts, Jo Tondeur, and Ronny Scherer, *A latent profile analysis of adult students' online self-regulation in blended learning environments*, Computers in Human Behavior **99** (2019), 126–136.

[248] Katrien Verbert, Erik Duval, Joris Klerkx, Sten Govaerts, and José Luis Santos, *Learning analytics dashboard applications*, American Behavioral Scientist **57** (2013), no. 10, 1500–1509.

[249] Arto Vihavainen, Matti Paksula, and Matti Luukkainen, *Extreme apprenticeship method in teaching programming for beginners*, Proceedings of the 42nd ACM technical symposium on Computer science education, 2011, pp. 93–98.

[250] Maria Virvou and Maria Moundridou, *A web-based authoring tool for algebra-related intelligent tutoring systems*, Journal of Educational Technology & Society **3** (2000), no. 2, 61–70.

[251] Kenneth Vollmar and Pete Sanderson, *Mars: an education-oriented mips assembly language simulator*, SIGCSE, vol. 6, 2006, pp. 239–243.

[252] Lev Semenovich Vygotsky, *Thought and language*, Annals of Dyslexia **14** (1964), no. 1, 97–98.

[253] Andrew Ward and Alan Jenkins, *The problems of learning and teaching in large classes*, Teaching large classes in higher education: How to maintain quality with reduced resources (1992), 23–36.

[254] Rasil Warnakulasooriya and David Pritchard, *Learning and problem-solving transfer between physics problems using web-based homework tutor*, EdMedia+ Innovate Learning, Association for the Advancement of Computing in Education (AACE), 2005, pp. 2976–2983.

[255] Evan Weingarten, Qijia Chen, Maxwell McAdams, Jessica Yi, Justin Hepler, and Dolores Albarracín, *From primed concepts to action: A meta-analysis of the behavioral effects of incidentally presented words.*, Psychological Bulletin **142** (2016), no. 5, 472.

[256] Peter S Westwood, *What teachers need to know about teaching methods*, Aust Council for Ed Research, 2008.

[257] David S White and Alison Le Cornu, *Visitors and residents: A new typology for online engagement*, First monday **16** (2011), no. 9.

[258] John Williams, *Stem education: Proceed with caution*, Design and Technology Education: An International Journal **16** (2011), no. 1, 1360–1431.

[259] Ben Williamson, *Digital education governance: Data visualization, predictive analytics, and 'real-time'policy instruments*, Journal of Education Policy **31** (2016), no. 2, 123–141.

[260] PH Winne and AF Hadwin, *Studying as self-regulated learning. dj hacker, j. dunlosky, ac graesser*, Metacognition in educational theory and practice (1998), 277–304.

[261] Alyssa Friend Wise, Yuting Zhao, and Simone Nicole Hausknecht, *Learning analytics for online discussions: Embedded and extracted approaches.*, Journal of Learning Analytics **1** (2014), no. 2, 48–71.

[262] Fiona Wright, David White, Tony Hirst, and Alan Cann, *Visitors and residents: Mapping student attitudes to academic use of social networks*, Learning, Media and Technology **39** (2014), no. 1, 126–141.

[263] Cheng Ye and Gautam Biswas, *Early prediction of student dropout and performance in moocs using higher granularity temporal information*, Journal of Learning Analytics **1** (2014), no. 3, 169–172.

[264] Pat Young, *'i might as well give up': Self-esteem and mature students' feelings about feedback on assignments*, Journal of Further and Higher education **24** (2000), no. 3, 409–418.

[265] Nevin Lianwen Zhang and David Poole, *A simple approach to bayesian network computations*, Proceedings of the Biennial Conference-Canadian Society for Computational Studies of Intelligence, CANADIAN INFORMATION PROCESSING SOCIETY, 1994, pp. 171–178.

[266] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan, *Large-scale parallel collaborative filtering for the netflix prize*, International conference on algorithmic applications in management, Springer, 2008, pp. 337–348.

[267] Barry J Zimmerman, *Becoming a self-regulated learner: Which are the key subprocesses?*, Contemporary educational psychology **11** (1986), no. 4, 307–313.

[268] Barry J Zimmerman and Anastasia Kitsantas, *Homework practices and academic achievement: The mediating role of self-efficacy and perceived responsibility beliefs*, Contemporary Educational Psychology **30** (2005), no. 4, 397–417.