




Constructive Alignment in Modern Computing Education: An Open-Source Computer-Based Examination System

Matthias Linhuber 
matthias.linhuber@tum.de
Technical University of Munich
Munich, Germany

Jan Philip Bernius 
janphilip.bernius@tum.de
Technical University of Munich
Munich, Germany

Stephan Krusche 
krusche@tum.de
Technical University of Munich
Munich, Germany

ABSTRACT

Large-scale paper-based examinations (PBEs) in computing education frequently emphasize rote memorization, thereby misaligning instructional objectives with assessment techniques. Such incongruities hinder the preparation of students for real world challenges in both industry and academia by inadequately evaluating higher-order cognitive abilities. Often, educators are deterred from implementing comprehensive skills assessment due to the perceived complexity and resource-intensive grading processes involved.

To mitigate these limitations, this paper introduces an exam mode as an integral feature of the open-source learning platform Artemis. Designed for both local and cloud-based deployment, this exam mode incorporates anti-cheating protocols, automates the grading of diverse exercise types, and features double-blind manual grading to ensure assessment integrity. It fosters the evaluation of complex cognitive skills while substantially reducing the administrative load on faculty.

This paper substantiates the effectiveness of the Artemis exam mode through widespread institutional adoption, demonstrated by over 50 successful computer-based examinations (CBEs). An in-depth case study involving 1,700 undergraduate software engineering students offers key insights, best practices, and lessons learned. This research not only pioneers the documentation of a secure, scalable, and reliable exam system at an institutional scale but also marks a seminal contribution to modernizing assessment strategies in computing education, with a particular focus on constructive alignment.




CCS CONCEPTS

• **Software and its engineering** → **Designing software**; • **Applied computing** → **Education**; • **Social and professional topics** → **Student assessment**.

KEYWORDS

Automated Assessment, Programming Exercises, Continuous Integration, Version Control, Instant Feedback, Online Editor, Plagiarism Checks, Open-Source Learning Platform, Exam Mode, Automated Grading, Scalability.

ACM Reference Format:

Matthias Linhuber , Jan Philip Bernius , and Stephan Krusche . 2023. Constructive Alignment in Modern Computing Education: An Open-Source Computer-Based Examination System. In *23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*, November 13–18, 2023, Koli, Finland. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3631802.3631818>

1 INTRODUCTION

The demand for computer scientists has witnessed substantial growth in recent years, resulting in large class sizes exceeding 1,000 students in undergraduate courses and 500 students in graduate courses. This presents a challenge for instructors, as individual interaction and customization of lectures, exercises, and assessments to heterogeneous student groups become impractical or even impossible. Instructors in such large courses encounter difficulties when assessing students. Many opt to prioritize lower cognitive skills to reduce grading efforts, as assessing exercises that involve repetition of previously learned concepts is comparatively easier. Assessing higher-order skills such as explanations, differentiations, or novel creative solutions to existing problems requires considerable effort. However, excluding exercises that require higher-level competencies from examinations may inadvertently promote rote memorization, leading to diminished learning outcomes in desired competencies.

The implementation of constructive alignment [4] offers a potential solution to the challenges faced by instructors [11]; however, its adoption remains challenging, as instructors often struggle to define learning outcomes and align learning activities with assessments. PBEs tend to restrict students to writing minimal code on paper, despite their acquisition of more complex programming skills using Integrated Development Environments (IDEs) throughout their studies. This exemplifies the inherent lack of constructive alignment using PBEs for assessing programming skills.

The COVID-19 pandemic necessitated a shift to online examinations due to distancing regulations, resulting in a rise in CBEs and the development of new solutions. However, with the relaxation of strict distancing rules, many instructors have reverted to supervised onsite PBEs, primarily due to concerns about increased cheating possibilities in online exams. While onsite supervised CBEs are feasible, apprehensions regarding the complexities of setup and associated risks deter their adoption. Ensuring sufficient power outlets, stable WiFi, and a reliable CBE system poses challenges, as universities often lack adequate computer rooms, necessitating students to use their own notebooks, which vary in operating systems. Consequently, guaranteeing security without intrusive measures on students' devices becomes impractical. This limits the available

online examination types to open book or open internet scenarios, where students can access external resources during the exam. Consequently, the range of exam questions that instructors can ask is restricted, and the complete reuse of previous exam questions becomes infeasible, as easy access to solutions in lecture slides or online sources would undermine the assessment of students' individual problem-solving competence.

These consequences require instructors to invest more time in creating problem statements. However, these challenges also have positive implications for teaching, as instructors are compelled to design original tasks with unique problems that closely resemble real world challenges faced by computer scientists in industry. Such exercises can effectively assess students' problem-solving skills by evaluating their ability to apply learned concepts within specific problem contexts. In addition to the advantages for students, instructors also benefit from CBEs. They offer automated grading, which reduces effort, enhances grading fairness, and shortens the time between examinations and result publication.

In this paper, we present the system Artemis that allows instructors to conduct large-scale online and onsite CBEs in a secure and reliable way. Section 2 discusses previous work in the area of computer-based assessment. In Section 3, we describe how the reference implementation Artemis supports quiz, text, modeling and programming assignments and their automatic correction in exams. Section 4 describes how Artemis implements the exam process with five distinct phases: design, preparation, conduction, correction, and review. Section 5 presents a case study of one large CBE with 1,700 registered students in a software engineering course. In Section 6, we present findings and lessons learned.

2 RELATED WORK

Online examination is a broad research field which has been of interest for many years. Yuan Zhenming et al. introduced a web-based exam system that supports objective questions and operating questions. They showed a framework to support auto grading of Microsoft Office submissions and other basic computer tasks [27]. Liang Zhang et al. introduced a web-based exam system in 2006 [21]. Part of their system is an auto grading component which helps to assess multiple choice, fill-in questions, and Microsoft Office submissions. Fluck et al. outlined an online examination strategy using a specialized live operating system [9]. The system can be configured for open and closed book exams. An instructor creates a document containing the exercises and students can use the provided software to work on them. Trivedi describes a similar setup, but uses a customized head set, and fingerprint readers as additional components [26]. The custom operating system ensures that only explicitly allowed computer aids can be accessed. All exam questions are delivered via an audio stream to the students headset. The accompanying desktop program shows a set of predefined answers the student can select. The system has several disadvantages since specialized hardware is required for all students, only multiple choice questions are supported, and the overhead introduced by the audio questions is significant. Fragulis et al. introduced a wordpress-based examination plugin which supports multiple choice and text submissions called ODES [10]. A student receives an auto generated,

random selection of questions based on constraints posed by the instructor. In addition, their system supports automatic grading.

In 2011, Terzis and Economides proposed a model which investigate factors for the acceptance of computer based assessment [25]. They found that the most important constructs are "perceived ease of use" and "perceived playfulness" for students. Piaw Chua showed that computer-based testing has better external and internal validity than paper-based testing methods [23]. The author found that computer-based testing results in better self-efficiency and higher intrinsic motivation. Boevé et al. found that CBEs do not influence the students scores in comparison to traditional exams in psychology, but the acceptance of such exams is lacking - especially because students are not familiar with the process [5]. Harley et al. showed that CBEs can reduce negative emotions in comparison to traditional examination scenarios [12]. The authors explicitly took the levels of anxiety, hopelessness, and shame into account. All three emotions are found to be lower in CBEs.

In a recent study, Eko et al. showed a fault-tolerant, web-based exam system targeted for developing countries [8]. The primary focus of this work was to show an exam system which is resilient to power losses and network issues. The authors implemented a web application which ensures exams can be resumed if disruptions occur. They support concurrent exams and random question selection. However, their system only supports multiple choice questions and has limited scaling potential due to the chosen data model.

The open-source community also implemented tools that support educators with grading. Moodle¹ is a broadly adopted Learning Management System (LMS) allowing educators to conduct exams. Ilias² is a similar open-source LMS that also supports online exams. However, both lack modeling and programming exercises support rendering them unusable for constructively aligned programming courses. JupyterHub combined with tools like nbgrader³ implements programming assignments and free text answers with automatic and manual grading workflows but lacks other assignment types like quizzes and modeling [13].

Also, commercial solutions are available. Inspera⁴ implements a cloud-based versatile assessment and certification workflow with 25 question types, including a basic support for programming assignments. Still, the current version is limited to small code snippets, and automatic assessment is not fully supported yet. Also, modeling support is missing. Similarly, Gradescope⁵ implements an exam platform with the addition of programming tasks. All commercial solutions presented here have the disadvantage that they are paid, and student data is managed outside of the university or school, which may lead to privacy issues.

Previous studies and the existing tool spectrum show that CBEs are desired. However, the existing solutions are limited regarding possible question types, automatic correction, price, ease of use, and acceptance. The open-source Artemis exam mode presented in this paper aims to solve the shown limitations by using the same familiar learning platform students use during the course for the final examination, providing diverse exercise types (quiz, modeling,

¹<https://moodle.org>

²<https://www.ilias.de/en>

³<https://nbgrader.readthedocs.io/en/stable>

⁴<https://www.inspera.com/product-overview>

⁵<https://www.gradescope.com>

text, and programming exercises), and corresponding automatic grading.

3 ARTEMIS AND EXAM MODE

Leveraging the open-source⁶ Learning Platform Artemis [16], the proposed exam mode enables instructors to curate exams for each course with customizable student registries and a diversity of exercise types. Artemis accommodates time extensions, augments reliability through local autosaving with 30-second synchronizations, thus circumventing potential issues due to internet instability.

Post-examination, the assessment dashboard facilitates instructors in visualizing and evaluating submissions, with options for multiple correction rounds. Following the assessments, Artemis provides the results online, granting students the opportunity to review and file for re-assessment, which engages a secondary reviewer for reconsideration. Artemis generates a comprehensive statistical report for performance analysis.

Artemis's unique functionality allows the creation of individualized exams, utilizing a set of variable exercises, randomized order, and a blend of compulsory and optional tasks, thereby augmenting the randomness of each assignment. Artemis offers robustness against internet disruptions, by permitting offline work on specific exercises and syncing saved progress upon connection recovery. For programming tasks, Artemis mandates internet connectivity only during solution submissions from their integrated development environment.

3.1 Exercise types

Artemis supports programming, modeling, quiz, text and file exercises. It utilizes version control and continuous integration for **programming exercises**, interfacing with corresponding servers for automatic assessment via test cases and static code analysis. This enables real-time, interactive feedback for students, while maintaining limited feedback during exams to prevent solution disclosure. Instructors populate a version control template repository with exercise skeleton code and dependencies, while auto-grading tests reside in a separate, student-inaccessible repository. Customizable combinations of black-box, white-box tests, and static analysis evaluate functionality, implementation details, and code quality.

The instructor sets a build plan in the Continuous Integration System (CIS), which compiles and tests the exercise code, including a task to pull updates from repositories and notify Artemis of results. Before an exam starts, Artemis produces personalized student repositories and build plans, which activate upon student code submissions, concealing continuous integration complexities from the student. For large-scale exams, Artemis efficiently manages a multitude of student-specific repositories and build plans.

Students can work either locally or via the online editor during exams. Upon submission, the build plan compiles, tests, and swiftly uploads results to Artemis for student review. Incorrect solutions provide error feedback, enabling multiple submission attempts. Instructors monitor progress in real time and can conduct manual reviews if needed, assessing aspects not covered by automatic tests or analysis.

⁶<https://github.com/ls1intum/Artemis>

Artemis also facilitates the composition of three **quiz** types: short-answer, drag-and-drop, and multiple-choice questions. For short-answer questions, instructors can control for typos and capitalization. Drag-and-drop questions can be created using images or modeling elements, with Artemis ensuring no unsolvable mappings. Multiple-choice questions, along with associated hints and explanations, are defined using markdown. Quizzes are auto-graded, random question and answer order can be implemented to deter collaboration. Various scoring strategies, including all or nothing, and proportional with or without penalty, are offered.

Artemis integrates an online **modeling** editor Apollon⁷ that is open-source. Apollon supports seven Unified Modeling Language (UML) diagrams: class, object, activity, use case, communication, component, and deployment diagrams. It also supports three additional non-UML diagram types: Petri nets, syntax trees, and flowcharts. Apollon is lightweight and easy to use to lower the entrance barrier of digital modeling.

3.2 Automatic Assessment

Artemis employs varying assessment strategies depending on exercise type. Programming exercises are auto-assessed using software tests and Static Code Analysis (SCA) rules, with instructor-determined weightings, hidden tests, and customizable SCA rules that measure code quality aspects. An optional manual assessment phase permits in-depth feedback provision. Quiz exercises are auto-assessed with scoring strategy-based results. Instructors can revise quizzes by adding text options or invalidating ambiguously formulated questions.

Artemis utilizes supervised machine learning for semi-automatic assessment of modeling and text exercises [2, 3, 15]. During manual assessment, Artemis learns correctness aspects and, through similarity analysis, proposes feedback for subsequent submissions. Exercise-specific feedback provision is enabled for different exercise types, with reviewer training implemented to ensure grading consistency and fairness.

The double-blind review process eliminates potential bias, and structured grading instructions further standardize grading. Pre-defined feedback can be applied via drag-and-drop mechanisms. A feedback rating system and a reviewer leaderboard foster high-quality feedback provision, aiding student comprehension and misconception prevention.

3.3 Architecture

Krusche and Seitz introduced the initial version of Artemis in 2018 [16]. The top-level architecture of Artemis consists of 5 subsystems. Artemis implements a client server architectural style with an *Artemis Application Server* and a *Artemis Application Client* connected via the *Artemis API*. The *Artemis Application Server* uses services provided by a *Version Control System*, a *Continuous Integration System*, and a *User Management System*. Figure 1 illustrates the connections between the subsystems. [16]

Since potentially hundreds of students access Artemis simultaneously during an exam, a scalable and highly available architecture is required. Both can be achieved by a vertical scaling approach. The Artemis is designed to support a clustered deployment based on

⁷<https://github.com/ls1intum/Apollon>

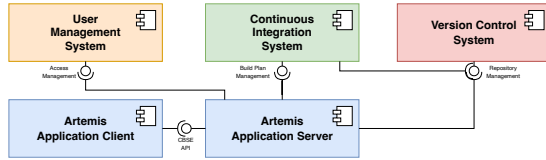


Figure 1: Artemis top level design with external subsystems

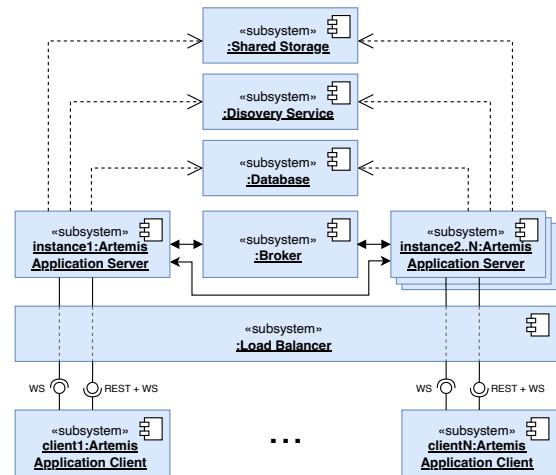


Figure 2: Artemis multi node architecture

multiple *Artemis Application Servers* as shown in Figure 2. A *Load Balancer* exposes the *Artemis Application Servers* to the *Artemis Application Clients* via *REST* and *Websockets (WS)* and handles the *TLS* termination. All nodes connect to a central *Database* and have access to a *Shared Storage* for persistent data management. In addition, we employ a *Message Broker* to distribute and relay *WS* requests, a *Shared Cache* to have common, distributed data structures between all nodes, and a *Discovery Service* to find and adopt nodes as they start up and add them to the cache pool.

This architecture allows dynamic scaling by adding or removing *Artemis Application Servers* on demand. The cluster automatically adopts new nodes and schedules new connections after they become ready. Artemis is highly available since the system continues to work as long as at least one *Artemis Application Server* remains functional.

4 EXAM PHASES

The Artemis exam mode is structured in five phases: design, preparation, conduction, correction, and review. Figure 3 shows all phases and the load distribution on the involved Artemis subsystems.

4.1 Design

During design, the instructors design the exercises they intend to use in the exam. Each exercise should be aligned with the course learning goals. The exercise types are of the same nature as the assignments during the semester. Hence, students have to use the same cognitive skills and strategies to get to a solution. Instructors use the workflow introduced by Krusche and Seitz to create exam exercises [16].

During creation of an exercise, an instructor writes a problem statement and a clear description of the action items the student has to solve. In addition, they create a sample solution (which may include assessment hints and details) and grading criteria which have to be met to get a specified number of points. For programming exercises, also software tests are required. Depending on the exam style, instructors may create tests which provide feedback during

the exam, or are only executed after the exam during the *Correction* phase.

Collaboration between students is a major concern in CBEs. With communication software being broadly available and digital exam solutions being easily shareable, it is very easy for students to exchange exam solutions among each other. To prevent this form of collaboration, every student gets their individual problem set. An exam typically consists of many exercises. For every exercise, instructors define one or more exercise variants. Artemis generates an individual set of assignments for every student by randomly selecting one variant for every exercise. Since variants are complete exercises on their own they can be completely unique and independent from each other. However, it is a good practice to limit variations to minor differences to keep the difficulty level comparable.

Finally, this phase includes quality control. Instructors typically review and test an exam before the conduction. This activity includes test runs by, e.g., colleagues which give feedback on the exercises and check the understandability of problem statements and the time constraints. The Artemis system supports this activity with its exam test run feature: instructors can compose an individual test exam based on the variants and invite the testers to execute the exam within the system. Thereby, the test runs can be configured so that all variants are covered and the tests can execute the exam in the actual examination system with a real working time.

4.2 Preparation

The first step in the exam preparation phase is to import all the students registered for the exam in to the system. This information may come from an external data source e.g. a central examination management system, or directly from the instructor. Each student receives an individual exam consisting of a permutation of exercises picked from the exercise variants the instructor created during the *Design* phase. The system ensures that all exercise variants are equally distributed. Instructors may adapt individual exams due

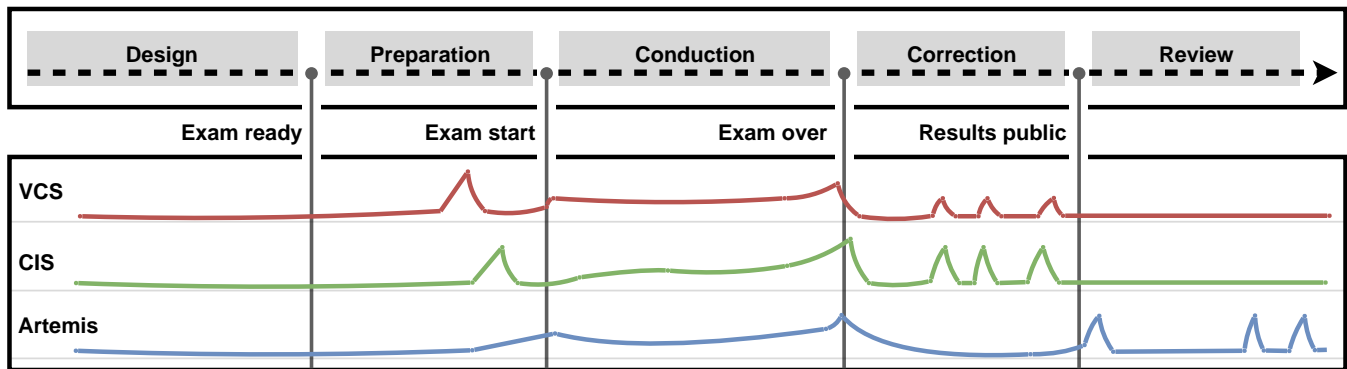


Figure 3: Exam conduction timeline with typical load patterns on the involved subsystems

to external constraints like extended working time or specialized exercise mappings.

Based on the individual exams, the system creates the corresponding exercise resources e.g. repositories and build plans. Since the creation of these resources requires considerable computing power, this activity is typically performed some time before the exam start. Instructors can monitor the generation of the exercise resources in the Artemis exam dashboard. By the end of the preparation phase, all individual student exams are composed and all resources are prepared for the exam conduction, enabling a smooth start of the exam. The primary load during this phase is on the Version Control System (VCS) and CIS due to the mass creation of exam resources.

4.3 Conduction

Five minutes before the official working time starts, students can open the exam view in Artemis and sign off exam hints, modalities, and terms. In this time frame, the *Artemis Application Client* downloads and caches the *Individual Exam* for the student in the background. The *Artemis Application Server* meanwhile adjusts the access permissions for all exam repositories in the VCS. The client ensures all exams start simultaneously by synchronizing a client-internal clock with the *Artemis Application Server*. This design decision distributes computationally intensive tasks to prevent a throughput bottleneck at the exam start.

As soon as the official working time starts, Artemis shows an overview of all exercises the student has to solve. The User Interface (UI) is closely aligned with the familiar exercise view students use to solve their homework. However, the UI is reduced to the necessary exam functionality to improve the visual affordance but have a consistent and standardized UI [22]. In addition, we show the remaining exam time in all views.

Instructors can choose whether students receive detailed feedback on their programming exercises during the exam, or only a basic compile check. Either way, students can submit programming exercises multiple times, Artemis builds the current state and shows feedback in the editor view. The feedback is generated asynchronously by the CIS and pushed to the *Artemis Application Client* via Websockets. The state in the client is automatically synced to the server every 30 seconds. In addition, Artemis allows to work

offline in case a student has connectivity issues. The state is synced to the server as soon as the connection is recovered.

At the end of the official working time, the students have to sign off and actively submit their exam. The instructor may configure a grace period to account for technical difficulties, and the time overhead introduced by git operations. The *Artemis Application Server* revokes access to the students' git repositories in the background. As this task takes some time to complete, Artemis additionally monitors late pushes and marks them as invalid. During this phase Artemis has a consisted, elevated load. The load on VCS and CIS is steadily rising and reaches a maximum at the final submission.

4.4 Correction

After the exam is completed, the correction phase starts. The system allows manual and automatic correction of exercises. Both are strictly double-blind and fully parallelizable. This is especially important for large exams with multiple instructors involved in the correction of exercises. Each exercise is equipped with a sample solution and initial grading criteria to support instructors to assess exams consistently. In addition, the system provides ML-Based grading and feedback suggestions [2, 3, 15]. There are cases in which grading criteria may change during the assessment, e.g. because the reviewers identify creative solutions that do not fit to the grading criteria, but still reflect a correct solution. The instructor can conduct multiple correction rounds to improve the quality of the correction in such situations.

Programming exercises are assessed by software tests and static code analysis. Initially, the system uses the tests instructors created during the *Design* phase. However, these tests may not be sufficient. Some initial software test may not cover important details which were forgotten in the *Design* phase. The system allows test adaption and automatic test reruns to re-evaluate and grade all submissions without manual intervention. This allows instructors to iterate over the grading criteria multiple times. Each re-evaluation shows the change of achieved points for each submissions. The primary load on the subsystems of Artemis is induced by these test-reruns.

In addition to the correction, Artemis performs automatic plagiarism detection for text, modeling and programming exercises. It analyzes the similarities between all student submissions and highlights those which exceed a given threshold. Artemis integrates

the open-source plagiarism tool JPlag [24] for programming and text exercises. For modeling exercises it offers a custom similarity assessment mechanism. With the integrated plagiarism editor, instructors can compare all highlighted submissions and confirm those actual plagiarism attempts cases. In addition, instructors can download a report of accepted and rejected plagiarism attempts. Artemis integrates functionality to notify the students, and allow them to react to the accusation.

4.5 Exam Review

Artemis offers the possibilities for online exam reviews so that students can participate in the review asynchronously reducing commute and time efforts. During the review period, students have the opportunity to evaluate the assessment of their exam in the web interface of Artemis. They can inspect the correction results and the feedback and compare their own solution with the example solution and the grading criteria provided by the instructor.

If they find inconsistencies, they can submit complaints about perceived mistakes triggering a re-evaluation of their exam exercise. The complaint review is triple-blind. Artemis ensures that a second reviewer evaluates the complaint who does not know neither the identity of the student nor the first reviewer. This increases the fairness and minimizes the risks of personal bias. Students may dispute plagiarism accusations identified in the *Correction* phase. After all complaints are reviewed, the final grades can be analyzed and exported to external systems.

5 CASE STUDY

This case study describes the instantiation of Artemis in the course Introduction to Software Engineering at Technical University of Munich (TUM). Traditionally, Introduction to Software Engineering is an on-campus lecture-based course taught to first-year bachelor computer science students. The course has grown rapidly over the last decade with 2,200 students registered for summer semester 2022. In summer 2020, the COVID-19 pandemic forced this course to be taught and examined in an online setting. As of the further increasing student numbers, we kept the examination online for the 2021 and 2022 editions of the course while teaching returned to an on-campus format in 2022. In this case study we investigate the following research questions by using the Artemis exam mode in Introduction to Software Engineering:

RQ1: What compute infrastructure is needed to host a reliable CBE system?

RQ2: Can CBEs reduce efforts in terms of design, conduction, and correction?

5.1 Course

The course covers a wide range of software engineering concepts, including requirements analysis, system and object design, testing, software lifecycles, configuration management, project management, and UML modeling [20]. Instructors establish specific learning objectives for each lecture based on the six cognitive skills outlined in Bloom's revised taxonomy [1]. The emphasis in this course is on higher cognitive skills, with students actively applying

the concepts through practical exercises. The teaching concepts and exercises are carefully aligned with the course objectives using the constructive alignment methodology [4].

Introduction to Software Engineering follows an interactive learning approach, employing multiple iterations of theory, examples, exercises, solutions, and reflection [17]. The course utilizes exercises to encourage student participation [18] and promote attendance in the lectures [19]. Throughout the semester, students engage in various types of exercises to prepare for the final exam, including in-class exercises during lectures, group exercises in small ad hoc groups, individual homework exercises, and team exercises over five 2-week sprints. To receive feedback and points, students are required to submit their exercise solutions to Artemis. Participation in the exercises also offers the opportunity to earn bonus points for the final exam. The assignments in the final exam follow the same concept and structure as the exercises during the semester.

5.2 Artemis Deployment at TUM

The deployment of Artemis for the case study is illustrated in Figure 4. The components of Artemis including the database, broker, shared storage, and load balancer are running on separate Virtual Machines (VMs). We scaled Artemis horizontally to 11 Artemis application server nodes, distributed across two classes of VMs. Specifically, six servers run on small VMs with 4 CPUs and 8 GB RAM, while five run on large VMs with 12 CPUs and 16 GB RAM.

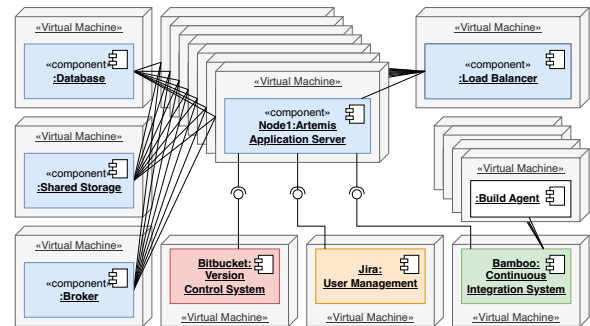


Figure 4: Artemis multi node deployment with external sub-systems at TUM

The Artemis instance at TUM uses Atlassian Jira as *User Management System*. Jira is connected to the universities central user directory. This architecture allows students to use their university account to log in to the systems while the instructors can manage access to different materials independent of the central authorities. Atlassian Bitbucket implements the *Version Control System*. It integrates well with the *User Management System* and can be scaled horizontally. Atlassian Bamboo is the *Continuous Integration System*. Bamboo delegates the build execution to 90 Linux *Build Agents* deployed as virtual machines to a Proxmox hypervisor clusters.

The *Discovery Service* is implemented by the JHipster Registry using Eureka. It ensures that the nodes find each other and establish the shared hazelcast cache. The Artemis system uses MySQL Server as *Database* implementation and ActiveMQ as *Broker* implementation. Nginx serves as the *Load Balancer* component. The system

uses the default load balancing strategy of nginx: weighted round robin. We found that for this use case round robin is a good load distribution strategy. In Section 5, we discuss a real world example and show the achieved distribution in more detail.

5.3 Introduction to Software Engineering Exam

In the following, we walk through the five exam phases introduced in Section 4 of the Introduction to Software Engineering exam from summer 2022.

Design: The exam consisted of eight exercises giving 100 points in total: four multiple-choice quiz exercises (20 points), one process modeling exercise (12 points), one design pattern modeling exercise (13 points) and two Java programming exercises on REST and Software Testing (25 and 30 points). For every exercise, we created between four and six exercise variants. Exercise variants provide a similar problem statement in terms of structure, but the instructors changed the underlying example so that the different variants test the same cognitive skill but pose a different problem context. For example, the design pattern exercise asks students to model an investment system showing current values based on the live price development on the stock market. Three possible variants in this example are: (1) a ETF investment system, (2) a single stock investment system, and (3) a crypto currency investment system.

Preparation: 1695 students were registered for the Introduction to Software Engineering exam. As part of the preparation phase, Artemis generates an individual exam for every student by choosing one variant for every exercise. Given the number of variants for the exercises, a total of 122,880 unique individual exams version exist. Figure 5 shows the distribution of variants for all eight exercises. For the two programming exercises, Artemis already creates git repositories containing the skeleton code, resulting in 3,390 individual repositories for the exam.

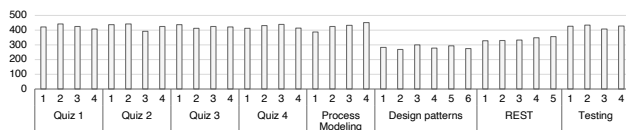


Figure 5: Introduction to Software Engineering exam with number of exercises, variants, and students per variant

Conduction: We performed the exam on an Artemis installation that is also used by other courses so the system operated in a typical production environment and was not limited or reserved for the exam conduction. Figure 6 depicts the number of active users / sessions in the Artemis system around the exam. 1,500 of the 1,695 registered students started the exam (88 %). 1,477 students handed in the exam in the end (98 %). We measured 2359 active users in the system 5 minutes before the exam start. During the exam period, the system peaked with 3163 active users around 20 minutes before the exam end. The exam ended with 2896 active users in the system. The number of active users declines quickly after the end of the exam with 2050 active users 5 minutes after the exam end. During the exam, we carefully monitored the system load of all machines in terms of CPU, RAM and disk IO usage. We did not record significant system load on the eleven Artemis node machines. However, we

monitored a slightly increased system load on the database server around the end of the exam. Figure 7 depicts the even distribution of users / sessions to the 11 nodes. Small instances (node{2-7}) host between 125 and 175 users each and large instances (node{1, 8-11}) host between 400 and 500 users.

We operated 90 build agents in the system of which 80 were capable to build and test student exam solutions. During the exam, the tests only verify whether the solution is compilable. Figure 8 depicts the number of active build agents over the course of the exam. Only 3 agents were active in the 5 minutes before the start of the exam. This metric slowly increases towards the end of the exam with 45 active agents 10 minutes before the end and all 80 capable agents active during the last 5 minutes of the exam.

Six people supervised the exam conduction: Three people monitored the compute infrastructure and another three people monitored the official chat room, looking out for problems and answering questions.

The Broker subsystem crashed between 15 and 10 minutes before the exam. This outage was detected by the monitoring system and could be resolved by an automatic system restart.

Correction: The exam correction process is different for the three different exercise types (quizzes, programming, and modeling) used in the exam. Multiple-choice quiz exercises can be evaluated automatically based on the solution mapping defined as part of the design phase. This evaluation has to be invoked manually by the instructor with the click of a button.

Programming exercises are corrected using a set of unit tests. The execution of the unit tests was scheduled to run after the exam between 08/09 00:00 and 08:00 grouped by exercise variant. These runs are visible as peaks in the build agent utilization in Figure 8. The instructors inspected the test results to adapt the preliminary test cases to correctly grade unexpected exam solution attempts. The test executions of the adapted tests can be seen in the following peaks in build agent utilization.

Modeling exercises were corrected manually. The assessment took two days with 15 - 20 humans grading in parallel. All grading was done in the Artemis web client using the grading criteria specified during the *Design* phase.

Review: Students were given 3.5 days to review their exams online in Artemis. In total, the instructors received 184 complaints on modeling exercises (manually graded) and 492 complaints on programming exercises (automatically graded using test cases). Students were aware that programming exercises were not reviewed by humans and argued how small mistakes might have affected more test cases to fail. The students did not have access to the instructors test cases, but received generated feedback messages.

5.4 Results

We answer the two research questions based on the case study in the Introduction to Software Engineering exam.

RQ1: The needed compute infrastructure primarily depends on the number of students involved. Two important measures are the number of Artemis server nodes and the number of build agents. We distinguish three scenarios:

- (1) Exams with **up to ~250 students**: a deployment with a single Artemis server node and five build agents is sufficient.

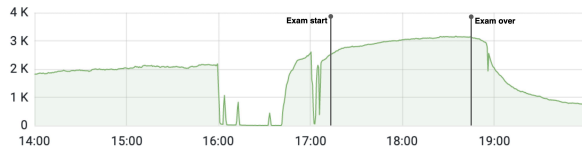


Figure 6: Artemis overall participating students

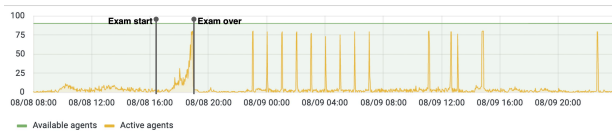


Figure 8: Build agent activity during and after the exam

- (2) Exams with **between ~250 and ~1,000 students**: vertical scaling of one Artemis server node with increased hardware resources in terms of CPUs and RAM is sufficient. For example, use a server with 12 CPU cores and 24 GB RAM when approaching the 1,000 student mark.
- (3) Exams with **more than ~1,000 students**: horizontal scaling of Artemis server nodes facilitates improved load distribution and ensures seamless failover mechanisms in the event of failures. To mitigate large build queues and ensure timely feedback within exams, it is necessary to utilize more than 20 build agents, as the capacity of these agents is a crucial determinant for prompt exercise feedback, directly impacting individual students.

Finding 1: One build agent per 50 participating students is a good estimate for efficient resource planning in CBEs with programming exercises.

RQ2: To conduct an effort comparison, this study examines the allocation of human resources and time between an online examination and the previous edition of the Introduction to Software Engineering course, which was administered in a traditional paper-based format in 2019 (pre-pandemic). Compared to the traditional PBE, several aspects of the online examination require additional efforts during the design phase:

- (1) **Open-book/Open-internet examination:** The online format necessitates the creation of different questions that target higher-level cognitive skills. Designing such questions is a time-consuming process, as they typically require contextualization to facilitate application. The replication of definitions is rendered irrelevant, as definitions or simple examples can be readily accessed from course materials or online sources.
- (2) **Creation of variants:** developing exercise variants amplifies the workload in exercise design, as the number of exercises increases accordingly. Although variants can often be derived from one another, meticulous consideration is essential to select a suitable set of examples.

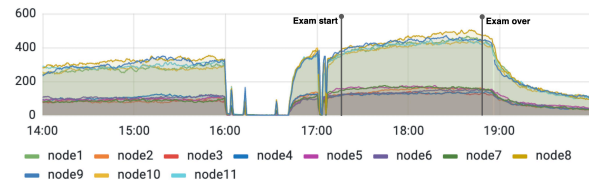


Figure 7: Artemis participating students per Artemis Node

- (3) **Automatic grading of programming exercises:** Setting up programming exercises for automatic grading necessitates the establishment of a comprehensive set of test cases. The definition of test cases is a time-consuming endeavor, particularly when accommodating different solution approaches is required.

The 2019 PBE was administered in ten lecture halls across two campuses in Munich. The conduction of the examination involved the participation of 50 university employees who performed tasks such as distributing and collecting exam sheets, verifying student identities, preventing cheating, and addressing questions. The allocation of staff was based on room capacities, where an average lecture hall was supervised by two scientific employees and three assistants.

In contrast, the CBE conducted in 2022 was remotely taken by students from their homes, obviating the need for lecture hall staff. The supervision of the CBE involved only six individuals. Consequently, the conduction effort was reduced by 88 %.

For the correction of the paper-based exam in 2019, we used an exam scanning solution which automated grading of multiple-choice exercises. Six exercises, including modeling, free text, and paper-based programming, had to be manually corrected. This manual correction process spanned ten days and an average of 24 people working eight hours per day in parallel, resulting in 1,920 correction hours. In contrast, the CBE only required two full working days with an average of 15 people involved, resulting in 240 hours. The manual correction efforts are decreased by 88 %.

The design of a CBE requires more initial setup effort compared to a PBE, especially in defining multiple variants for deception prevention and test cases for automatic evaluation of programming exercises. However, this effort is more than compensated by significant savings during execution and correction.

Finding 2: The combined effort of preparing, conducting, and correcting a CBE is significantly less than a PBE when more than ~100 students participate.

Finding 3: CBEs can be realized at large scale while involving less people.

6 DISCUSSION AND LESSONS LEARNED

CBEs have several advantages over PBEs, but also introduce challenges and risks. We discuss the most important aspects and present the lessons learned in the following section.

6.1 Exercises

CBEs necessitate exercises that target cognitive skills different from those assessed in traditional paper-based exams. It is crucial to prevent students from easily finding solutions to problem statements online. Hence, instructors may be tempted to create more challenging programming exercises, assuming that the IDE provide sufficient assistance to students for simple aspects (e.g., creating getters and setters), thus justifying more complex problem statements. However, given the limited time and stressful nature of exams, instructors should carefully estimate the time required for students to read the problem statement, comprehend the template code, and solve the exercise.

To provide a more realistic estimate of the time students will need to complete the task, instructors should calculate the difference/delta between the solution and template code in programming exercises by quantifying the number of lines students need to add and modify. These lines can be categorized into simple code (e.g., imports, getters, setters), moderately heavy code (e.g., basic logic statements), and heavy code (e.g., complex logic statements), and different time estimates can be assigned accordingly. It is also beneficial to grant additional working time (e.g. 10 minutes) to account for various technical issues and overhead that may arise during code cloning and uploading.

Finding 4: To optimize the effectiveness of programming exercises, it is essential to incorporate contextual problem statements that align with the learning objectives, while also categorizing the solution code based on its complexity to facilitate accurate estimation of the required working time.

6.2 Variants

Similarly, exercise variants should not differ significantly to ensure comparability of results. An effective strategy is to create a complete exercise variant with a specific context in a particular application domain. Subsequently, instructors can generate variants by altering the application domain. For example, one exercise variant could be "Implement a product API using REST for a shop selling mobile phones," while another variant could be "Implement a product API using REST for a shop selling cereal."

Finding 5: The implementation of a baseline exercise within a specific context, followed by the adaptation of the exercise to other domains, represents a fair strategy for generating comparable exercise variants.

6.3 Live Feedback

Providing detailed feedback during exams poses challenges and may not be desirable for several reasons. Firstly, it encourages students to share solutions with their peers if they receive a 100 % score. Secondly, it becomes difficult to modify grading criteria after the fact, as students may cease working on the exercise once they receive the 100 % solved feedback even though the solution is not fully implemented. Lastly, if there is a mistake in the tests and a student does not receive a success message during the exam due to an error, they may complain about the time lost. These risks and issues make live feedback undesirable for most use cases. However, it can be helpful to inform students if their own code

would successfully compile against the predefined tests in the CIS environment.

Finding 6: Feedback for programming tasks during the CBE beyond a compilation check is **not** recommended.

6.4 Acceptance

According to our observations, the acceptance among students is high. Most students prefer CBEs to PBEs because they can use their familiar programming tools instead of programming on paper, and because the scenario is more realistic. However, a few students complain that the CBEs are more difficult than the traditional exams. We assume that this complaint is independent of the CBE and is more related to the fact that constructively aligned assessments focus more on higher cognitive skills instead of pure knowledge questions. Hence, rote learning is not sufficient to pass the exam and students cannot efficiently pass a course with one day of effort anymore. We aim to quantify this in future research.

Creating open-book exams is more difficult and time consuming than creating closed-book exams since straightforward knowledge questions should not be used. The same applies for CBEs. Instructors have to invest time and effort into creating good and constructively aligned exams while keeping the difficulty level reasonable. By using automatic assessment, instructors can make up this time later in the process, which would lead to a generally high acceptance among them. However, due to increased cheating possibilities, some instructors fear academic integrity loss and still prefer PBEs, in particular if they can delegate the assessment to doctoral students or teaching assistants.

Finding 7: Most students prefer CBEs, while there are mixed feelings on the instructor side.

6.5 Security

A CBE setup needs to be managed with great care and attention to detail. The introduced overhead of managing such deployments by hand is significantly and error-prone. A rigorous configuration management solution should be used to manage these systems automatically. The Artemis Ansible⁸ collection installs, configures, and updates all components in a declarative fashion. This approach not only ensures a consistent and up-to-date production setup, but also canonical production-equivalent test setups.

Confidentiality, authenticity, and data integrity are crucial security design goals for automatic assessment in CIS. First and foremost, students may submit arbitrary code to the VCS which is built and tested automatically. The tests imply a remote code execution of the student code, which needs to be secured to avoid issues. Depending on the programming language and test strategy, this may lead to substantial security risks. Docker containers isolate the build execution from the host operating system and other build executions. They provide individual execution environments for different programming languages. The careful creations of tests with timeouts and strict security are important.

System availability is a central concern in exams. Infrastructure forms an interesting target for Denial of Service (DoS) attacks. While it is hard to prevent these attacks completely, administrators

⁸<https://www.ansible.com>

can make them more difficult and reduce the impact as much as possible. Simple measurements include adding rate limits to costly API endpoints in the load balancing component and setting up tools like fail2ban to block malicious IP addresses automatically.

Finding 8: Providing a secure and reliable exam infrastructure at scale requires substantial effort.

6.6 Deception

CBEs at home are more susceptible to deception than CBEs and PBEs in the lecture hall. In recent years, large language models became more and more relevant. Significant advancements in AI, such as GPT-3, GPT-4 and their general availability via ChatGPT⁹ influence education [14]. Traditional text-based questions are no longer sufficient to assess students [7]. Even solutions for tasks where the student is supposed to argue in a certain context can now be generated automatically to some extent. The exam in Section 5 did not include text exercises for this reason. Also, programming exercises are susceptible to AI generated solutions from tools such as ChatGPT, Codex¹⁰, and Github Copilot¹¹. A remedy to this problem are supervised CBEs in the lecture hall with computers provided by the university. An alternative for larger courses would be scenarios in which students bring their own device. While the effort for the conduction of the exam would increase again, the presented benefits in terms of automatic assessment and constructive alignment still hold. Due to the supervision in the lecture hall, deception would be less likely.

7 CONTRIBUTIONS AND FUTURE WORK

This paper presents three main contributions: first, it shows the design and architecture for a scalable CBE system that enables instructors to conduct constructively aligned, personalized exams. Artemis supports instructors with semiautomatic assessment for quiz, modeling, text, and programming exercises, parallel double-blind manual correction and triple-blind student complaints. Second, it formalizes a scaled exam conduction process consisting of five phases: Design, Preparation, Conduction, Correction, and Review. Third, it presents eight findings from a case study with 1,700 students that shows the use of the Artemis exam mode in a large software engineering exam.

The presented research can be extended in the following ways:

Evaluation across institutions: The Artemis system was successfully used for examination in multiple higher education institutions, especially during the COVID-19 pandemic. However, no formal evaluation was conducted across institutions. Future research should identify patterns and best practices of CBEs and their infrastructure.

Bring your own device (BYOD) exams: The focus of this paper primarily revolves around CBEs conducted online. Nevertheless, onsite CBEs also present a viable alternative for assessing students, depending on the availability of adequate computer labs. Considering the limited availability of computer rooms in most universities, the adoption of BYOD exams, where students use their own laptops in exam rooms, emerges as a logical next step. However, BYOD

exams introduce new challenges related to onsite infrastructure, including power supply and internet connectivity. Further investigation is warranted to analyze the challenges, benefits, and risks associated with BYOD exams.

AI in the examination: The effect of AI, especially large language models such as ChatGPT, is a current topic in education research [6, 14]. Future investigations should rigorously explore the implications of AI in exam environments. This includes the development of technical methodologies for identifying AI-generated submissions and ethical inquiries into the responsible utilization of AI technologies by both students and instructors. AI has the potential to help instructors generate variants or complete exercises during the *Design* phase, reducing the effort for large scale exams.

Dynamic application scaling: In this study, the system was scaled manually. A transition from a VM-based deployment approach to a container-based approach may be evaluated for exam use cases. Future work is needed to explore the potential uses for microservice architectures in CBE.

In conclusion, this research not only advances the field of CBEs through its innovative design and comprehensive evaluation but also provides actionable insights for educators and institutions aiming for more effective, scalable, and constructively aligned assessments. As computing education continues to evolve, the contributions of this paper lay a critical foundation for future empirical studies and technological advancements in assessment strategies.

REFERENCES

- [1] Lorin W. Anderson, David R. Krathwohl, Peter W. Airasian, Kathleen A. Cruikshank, Richard E. Mayer, Paul R. Pintrich, James Rath, and Merlin C. Wittrock. 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longmans Green.
- [2] Jan Philip Bernius, Stephan Krusche, and Bernd Bruegge. 2021. A Machine Learning Approach for Suggesting Feedback in Textual Exercises in Large Courses. In *8th ACM Conference on Learning @ Scale (Potsdam, Germany) (L@S '21)*. 173–182. <https://doi.org/10.1145/3430895.3460135>
- [3] Jan Philip Bernius, Stephan Krusche, and Bernd Bruegge. 2022. Machine learning based feedback on textual student answers in large courses. *Computers and Education: Artificial Intelligence* 3 (2022), 100081. <https://doi.org/10.1016/j.caeai.2022.100081>
- [4] John Biggs. 2003. Aligning teaching and assessing to course objectives. *Teaching and learning in higher education: New trends and innovations* 2 (2003), 13–17.
- [5] Anja J. Boevé, Rob R. Meijer, Casper J. Albers, Yta Beetsma, and Roel J. Bosker. 2015. Introducing Computer-Based Testing in High-Stakes Exams in Higher Education: Results of a Field Experiment. *PLoS ONE* 10, 12 (Dec. 2015), e0143616. <https://doi.org/10.1371/journal.pone.0143616>
- [6] Thomas K.F. Chiu, Qi Xia, Xinyan Zhou, Ching Sing Chai, and Miaoting Cheng. 2023. Systematic literature review on opportunities, challenges, and future research recommendations of artificial intelligence in education. *Computers and Education: Artificial Intelligence* 4 (2023), 100118. <https://doi.org/10.1016/j.caeai.2022.100118>
- [7] Debby Cotton, Peter Cotton, and J. Reuben Shipway. 2023. *Chatting and Cheating. Ensuring Academic Integrity in the Era of ChatGPT*. Preprint. EdArXiv. <https://doi.org/10.35542/osf.io/mrz8h>
- [8] Ceasar E. Eko, Idongesit E. Eteng, and Eyo E. Essien. 2022. Design and Implementation of a Fault Tolerant Web-Based Examination System for Developing Countries. *Eastern-European Journal of Enterprise Technologies* 1, 2(115) (Feb. 2022), 58–67. <https://doi.org/10.15587/1729-4061.2022.253146>
- [9] Andrew Fluck, Darren Pullen, and Colleen Harper. 2009. Case Study of a Computer Based Examination System. *Australasian Journal of Educational Technology* 25, 4 (Sept. 2009). <https://doi.org/10.14742/ajet.1126>
- [10] George F. Fragulis, Lazaros Lazaridis, Maria Papatismouli, and Ioannis A. Skordas. 2018. O.D.E.S.: An Online Dynamic Examination System Based on a CMS Wordpress Plugin. In *2018 South-Eastern European Design Automation, Computer Engineering, Computer Networks and Society Media Conference (SEEDA_CECNSM)*. IEEE, Kastoria, 1–8. <https://doi.org/10.23919/SEEDA-CECNSM.2018.8544928>
- [11] Telle Hailikari, Viivi Virtanen, Marjo Vesalainen, and Liisa Postareff. 2021. Student perspectives on how different elements of constructive alignment support active learning. *Active Learning in Higher Education* (2021).

⁹<https://chat.openai.com>

¹⁰<https://platform.openai.com/docs/guides/code>

¹¹<https://github.com/features/copilot>

- [12] Jason M. Harley, Nigel Mantou Lou, Yang Liu, Maria Cutumisu, Lia M. Daniels, Jacqueline P. Leighton, and Lindsey Nadon. 2021. University Students' Negative Emotions in a Computer-Based Examination: The Roles of Trait Test-Emotion, Prior Test-Taking Methods and Gender. *Assessment & Evaluation in Higher Education* 46, 6 (2021), 956–972. <https://doi.org/10.1080/02602938.2020.1836123>
- [13] Project Jupyter, Douglas Blank, David Bourgin, Alexander Brown, Matthias Bussonnier, Jonathan Frederic, Brian Granger, Thomas Griffiths, Jessica Hamrick, Kyle Kelley, M Pacer, Logan Page, Fernando Pérez, Benjamin Ragan-Kelley, Jordan Suchow, and Carol Willing. 2019. Nbgrader: A Tool for Creating and Grading Assignments in the Jupyter Notebook. *Journal of Open Source Education* 2, 11 (Jan. 2019), 32. <https://doi.org/10.21105/jose.00032>
- [14] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günemann, Eyke Hüllermeier, Stephan Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Aleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, and Gjergji Kasneci. 2023. *ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education*. Preprint. EdArXiv. <https://doi.org/10.35542/osf.io/5er8f>
- [15] Stephan Krusche. 2022. Semi-Automatic Assessment of Modeling Exercises Using Supervised Machine Learning. In *Hawaii International Conference on System Sciences*. <https://doi.org/10.24251/HICSS.2022.108>
- [16] Stephan Krusche and Andreas Seitz. 2018. ArTEMiS: An Automatic Assessment Management System for Interactive Learning. In *49th ACM Technical Symposium on Computer Science Education (Baltimore, Maryland, USA) (SIGCSE '18)*. 284–289. <https://doi.org/10.1145/3159450.3159602>
- [17] Stephan Krusche and Andreas Seitz. 2019. Increasing the Interactivity in Software Engineering MOOCs - A Case Study. In *52nd Hawaii International Conference on System Sciences*. 1–10.
- [18] Stephan Krusche, Andreas Seitz, Jürgen Börstler, and Bernd Bruegge. 2017. Interactive learning: Increasing student participation through shorter exercise cycles. In *19th Australasian Computing Education Conference*. ACM, 17–26.
- [19] Stephan Krusche, Nadine von Frankenberg, and Sami Afifi. 2017. Experiences of a Software Engineering Course based on Interactive Learning. In *Tagungsband des 15. Workshops Software Engineering im Unterricht der Hochschulen (SEUH)*. CEUR, 32–40.
- [20] Stephan Krusche, Nadine von Frankenberg, Lara Marie Reimer, and Bernd Bruegge. 2020. An interactive learning method to engage students in modeling. In *International Conference on Software Engineering: Software Engineering Education and Training*. 12–22.
- [21] Liang Zhang, Yue-ting Zhuang, Zhen-ming Yuan, and Guo-hua Zhan. 2006. A Web-Based Examination and Evaluation System for Computer Education. In *Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06)*. IEEE, Kerkrade, The Netherlands, 120–124. <https://doi.org/10.1109/ICALT.2006.1652383>
- [22] J. Nielsen. 1994. *Usability Engineering*. Elsevier Science.
- [23] Yan Piaw Chua. 2012. Effects of Computer-Based Testing on Test Performance and Testing Motivation. *Computers in Human Behavior* 28, 5 (2012), 1580–1586. <https://doi.org/10.1016/j.chb.2012.03.020>
- [24] Lutz Prechelt, Guido Malpohl, Michael Philippsen, et al. 2002. Finding plagiarisms among a set of programs with JPlag. *J. Univers. Comput. Sci.* 8, 11 (2002), 1016.
- [25] Vasileios Terzis and Anastasios A. Economides. 2011. The Acceptance and Use of Computer Based Assessment. *Computers & Education* 56, 4 (May 2011), 1032–1044. <https://doi.org/10.1016/j.compedu.2010.11.017>
- [26] Aakash Trivedi. 2010. A Relevant Online Examination System. In *2010 International Conference on Technology for Education*. IEEE, Mumbai, India, 32–35. <https://doi.org/10.1109/T4E.2010.5550114>
- [27] Yuan Zhenming, Zhang Liang, and Zhan Guohua. 2003. A Novel Web-Based Online Examination System for Computer Science Education. In *33rd Annual Frontiers in Education, 2003. FIE 2003*, Vol. 3. IEEE, Westminster, Colorado, USA. <https://doi.org/10.1109/FIE.2003.1265999>