ArTEMiS - An Automatic Assessment Management System for Interactive Learning

Stephan Krusche
Technische Universität München
Munich, Germany
krusche@in.tum.de

Andreas Seitz
Technische Universität München
Munich, Germany
seitz@in.tum.de

ABSTRACT

The increasing number of students in computer science courses leads to high efforts in manual assessment of exercises. Existing assessment systems are not designed for exercises with immediate feedback in large classes. In this paper, we present an **A**u**T**omated ass**E**ssment **M**anagement **S**ystem for interactive learning.

ArTEMiS assesses solutions to programming exercises automatically and provides instant feedback so that students can iteratively solve the exercise. It is open source and highly scalable based on version control, regression testing and continuous integration. ArTEMiS offers an online code editor with interactive exercise instructions, is programming language independent and applicable to a variety of computer science courses. By using it, students gain experiences in version control, dependency management and continuous integration.

We used ArTEMiS in 3 university and 1 online courses and report about our experiences. We figured out that ArTEMiS is suitable for beginners, helps students to realize their progress and to gradually improve their solutions. It reduces the effort of instructors and enhances the learning experience of students.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; *Computer science education*; • **Applied computing** → **Interactive learning environments**; **Learning management systems**;

KEYWORDS

Automated Assessment, Programming Exercises, Continuous Integration, Version Control, Instant Feedback, Online Editor, Interactive Exercise Instructions, Online Courses, In-class Exercises.

ACM Reference Format:
Stephan Krusche and Andreas Seitz. 2018. ArTEMiS - An Automatic Assessment Management System for Interactive Learning. In *SIGCSE '18: SIGCSE '18: The 49th ACM Technical Symposium on Computing Science Education, February 21–24, 2018, Baltimore, MD, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3159450.3159602

SIGCSE '18, February 21–24, 2018, Baltimore, MD, USA
2018. ACM ISBN 978-1-4503-5103-4/18/02…$15.00
https://doi.org/10.1145/3159450.3159602

1 INTRODUCTION

The amount of students in university classes and online courses is increasing. The number of freshmen at our computer science department increased by 67 % between 2013 (1110) and 2016 (1840). In addition, the number of enrollments in online courses such as MOOCs is increasing as well. Class Central recently reported 35 million students which have signed up for at least one MOOC. With such large numbers of students, manual assessment of programming exercises in computer science and software engineering courses is no longer feasible. Yet, programming exercises are essential in computer science education [17].

While automatic assessment is an established concept, large courses raise new challenges for instructors and tools. Automated assessment must be scalable to handle a large number of students and provide immediate feedback. The effort for an instructor conducting an programming exercise has to be independent of the number of participating students. Instant feedback to students is required to allow resubmissions and learning from failures. This is particularly important in courses based on active learning [3], where the programming exercises happen in-class in specific time frames. During the exercise, instructors need to be able to get an overview of submitted solutions and typical problems to react and guide the students.

There are many existing tools for automatic assessment and grading but most of them are custom tailored solutions for specific programming languages and requirements (compare Section 5). This makes it particularly hard to integrate them into existing infrastructure and use them in large scale courses.

Continuous integration (CI) is an approach allowing to detect defects and failures in programs [7]. This idea can be used for programming exercises to validate the correctness and completeness of source code with test cases. Students upload their solution to a version control system (VCS) and receive instant feedback about the result of their submissions from the CI server. While doing the exercise, they get used to VC and CI, which are important skills in software development.

In this paper, we describe our experiences with ArTEMiS, an open source **A**u**T**omated ass**E**ssment **M**anagement **S**ystem for interactive learning. In Section 2, we cover the basic background behind the methodology and tools for our approach. Section 3 presents the approach behind and the use of ArTEMiS in more detail. We describe its scalability and show the applicability of the system within large university courses and an online course in a multi case study in Section 4. We analyze the participation of students in the case study and provide insights. Section 5 relates and differentiates ArTEMiS to other automated assessment tools. In Section 6, we conclude the paper and provide directions for future work.

2 FOUNDATIONS

ArTEMiS uses version control (VC) and continuous integration (CI) to automatically assess programming exercises in interactive learning environments. This sections describes its foundations.

2.1 Interactive Learning

Interactive learning combines lectures and exercises into interactive classes with multiple iterations of theory, example, exercise, solution and reflection [12]. Educators teach and exercise small chunks of knowledge in short cycles. They focus on immediate feedback to exercises to improve the learning experience in large classes so that students reflect and increase their knowledge incrementally.

Hands-on activities in class increase students' motivation and engagement and allow continuous assessment over the course [13]. This approach expects active participation of learners and the use of computers (laptops, tablets, smartphones) in the classroom. Instructors provide guidance during the learning process to prevent misconceptions and to facilitate the learning process.

2.2 Continuous Integration

CI was first described by Grady Booch as concept to avoid risky late integrations [4]. Martin Fowler defines CI as follows: "Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily – leading to multiple integrations per day" [7].

Each developer works on a local copy of a shared code base. After implementing source code, the developer integrates the changes into the shared code base. An automated build (including test cases) verifies each integration to detect compile, test, and integration errors as soon as possible [7]. The build is triggered on the developer's machine or on a central CI server. While CI does not require special tools, developers use dedicated servers to perform it.

Figure 1 illustrates a common CI workflow. (1) The process starts with a developer committing code changes to a VCS. (2) The CI server regularly checks the VCS for code changes and (3) automatically triggers the build and test process on every commit. Tests can range from small unit tests over larger integration tests to complete system tests. (4) After the build has either succeeded or failed, the developer is notified about the build result. Every commit triggers a build, so the developer responsible for the failure is notified immediately to fix the problem.

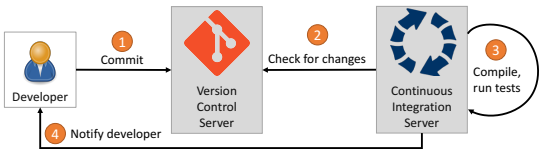

Figure 1: Typical continuous integration workflow

2.3 Automatic Assessment

"The [manual] assessment of [programming] assignments places significant demands on the instructor's time and other resources" [5]. To solve this problem and due to the logical character of programming, the task of assessing programming exercises can be automated. A first example was a grading program for punch card programs by Hollingsworth [9]. Since then, many automatic assessment tools (e.g. [6], [11], [15]) have been developed, together with guidelines on how programming exercises should be assessed.

Automatic assessment systems provide feedback on students' solutions for programming exercises [19]. Compared to manual assessment, they are able to provide consistent feedback for every student without bias while significantly reducing the effort for instructors and TAs. Ala-Mutka describes different types of assessments [1]. The historically most common type is dynamic assessment. Its main aspects are to assess functionality, efficiency, and testing skills by executing a program with test input data and checking the output for correctness. Another type is static analysis, which provides feedback on style, programming errors or software metrics by analyzing the code without executing it.

Students using automatic assessment tools achieve important learning goals: they develop a clean and reusable code style, reflect critically on errors and establish a testing culture. These goals are achieved by multiple methods, e.g. varying the amount of feedback provided or allowing to work incrementally on a solution. They can be effectively implemented using different assessment techniques such as black-box testing, white-box testing or peer reviews.

3 APPROACH

We identified the following goals for ArTEMiS by analyzing the applicability of existing tools for large interactive courses:

Independence of programming language: different programming languages are taught in university courses [17]. The system should work independently of a specific programming language.

Scalability: the system should be scalable usable in university classes and online courses. It should work with hundreds of participating students at the same time, e.g. during an interactive class. The workload for preparing and grading exercises must be independent of the number of participating students.

Instant feedback: the system should provide instant feedback for submitted solutions and describe why the particular solution is correct or wrong so that students can improve it.

Learning from failures: students can learn, reflect and iteratively submit new solutions even if they fail initially. Students should have the chance to resubmit their solutions as often as they want (potentially limited to a specific time frame).

Different exercise types: the system allows programming exercises in different areas of the software engineering process including programming basics, system design, object design, testing and build and release management.

Different assessment ways: the system should allow different ways to assess submissions. For programming exercises the system has to support different test types (e.g. structural, behavioral, runtime, performance or functional tests).

Traceability: the system must ensure traceability for instructors and for students. With regards to team exercises, the contribution of each team member should be accountable. Traceability enables early detection of difficulties for the students in the assignment and lets the instructor react accordingly.

Immediate evaluation: solutions and results are easily accessible and evaluable for instructors enabling them to remove ambiguity, answer open questions, or extend the given working time.

Interactive exercise instructions: dynamic tasks and UML diagrams visualize the current progress of students. They update their color from red (incomplete) to green (complete) when students submit their solution and when associated test cases pass.

Easy to use online editor: to simplify the participation and improve the learnability, programming beginners can work on programming exercises in an interactive and lightweight online editor. They can submit their solutions with just one button click.

ArTEMiS fulfills these goals by using the concepts of VC and CI. Each student works with a given template code in his own repository and has a build plan which executes test cases after each commit. Students can solve the programming exercise in an online editor, on the local computer using an IDE or with a mix of both. When using the online editor, they don't need to setup a VC client and an IDE. If they work on their own computer, they need to apply version control and install an IDE (e.g. Eclipse) and have more functionalities in the code editor (e.g. auto completion, error highlighting, etc.). Conducting a programming exercise consists of 7 steps distributed among instructor, ArTEMiS and students:

1. **Instructor prepares exercise:** set up a base repository containing the exercise code and test cases. Set up a base build plan on the CI server, and create the exercise on ArTEMiS.
2. **Student starts exercise:** click on *Start Exercise* on ArTEMiS. This automatically generates a copy (fork) of the base repository with the exercise code and a copy of the base build plan. ArTEMiS sets the permissions so that students can only see their personal repository.
3. **Student clones repository:** optionally clone the personalized repository from the remote VCS to the local machine.
4. **Student solves exercise:** Solve the exercise with an IDE of choice on the local computer **or** in the online editor.
5. **Student submits solution:** upload source code changes to the VCS by committing and pushing them to the remote server **or** by clicking *Commit & Run Tests* in the online editor.
6. **CI server verifies solution:** verify the student's submission by executing the test cases (see step 1) and provide feedback which parts are correct or wrong.

7a. **Instructor reviews course results:** review overall results of all students, and react to common errors and problems.

7b. **Student reviews personal result:** review build result and feedback using ArTEMiS. In case of a failed build, reattempt to solve the exercise (step 4.).

Figure 2 shows this approach as UML activity diagram. CI server, VCS, and ArTEMiS are combined in the *System* actor. The approach consists of 2 phases, exercise preparation and exercise execution.

Exercise preparation: an instructor sets up a VC repository containing the exercise code (template) handed out to students and test cases to verify students' submissions (*base repository*). This repository typically includes a small sample project including some predefined classes, dependencies to external libraries, e.g. a testing framework, and test cases. A combination of behavioral (black-box) and structural (white-box) tests allows to check for both functionality and implementation details of the submitted code. In addition, the instructor stores the tests separately in a test repository, which is not accessible to students, to prevent that they adapt the test cases. It can make sense to completely hide the tests from the students to prevent reverse engineering the solution.

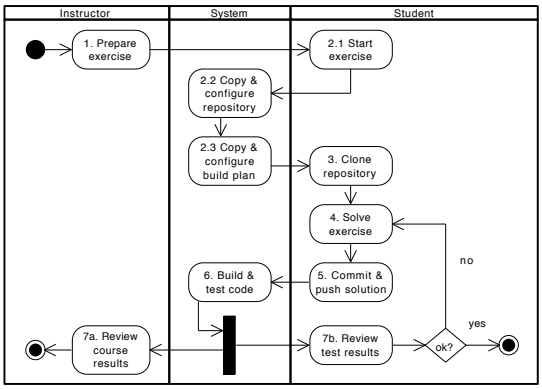

Figure 2: Process for automated assessment in ArTEMiS

After setting up the base and test repositories, the instructor configures the build plan on the CI server which compiles and tests the exercise code using the previously defined test cases (*base build plan*). This build plan includes a task to pull the source code from the base repository and the test repository whenever changes occur, and to combine them so that the tests can be executed in the second step. A final task, which is also executed when compilation or test execution fails, notifies ArTEMiS about the new result. The instructor finally creates an exercise on ArTEMiS by selecting the preconfigured base repository of the VCS and the preconfigured base build plan of the CI server.

Exercise execution: a student starts an exercise with a single click, triggering the setup process: ArTEMiS creates a personal copy of the base repository, the *student repository*, and provides access only to this student. It creates a personal copy of the base build plan, the *student build plan*, and configures it to be triggered when the particular student uploads changes to this personal student repository. The student can usually not access the build plan to hide its complexity. Personalized means that each student gets one repository and one build plan. When 200 students participate in an exercise, ArTEMiS creates 200 student repositories and 200 student build plans. Students only have access to their personal repository, they cannot access other student repositories. This prevents cheating, because students cannot access code of each other.

After the setup is complete, ArTEMiS displays the clone URL and/or allows the student to open the exercise in the online editor. The student clones the repository to the local computer and starts working on the exercise. When the students uploads a new solution to the personalized repository, the personalized build plan of the student assesses the solution. Students upload solutions by committing and pushing changes in their source code to their personal repository or by submitting their changes in the online editor (which triggers a commit and a push operation in the background). The new commit on the personal repository triggers the personal build plan to assess the solution on a build agent. The build agent pulls the submitted code from the personal repository and the tests from the test repository, and combines them in a working directory. It compiles the code, executes the tests and uploads the results to ArTEMiS in a few seconds, so that the student can immediately review the feedback and iteratively improve their solution.

In case of an incorrect solution, the feedback includes how many tests failed and the corresponding failure message for each failed

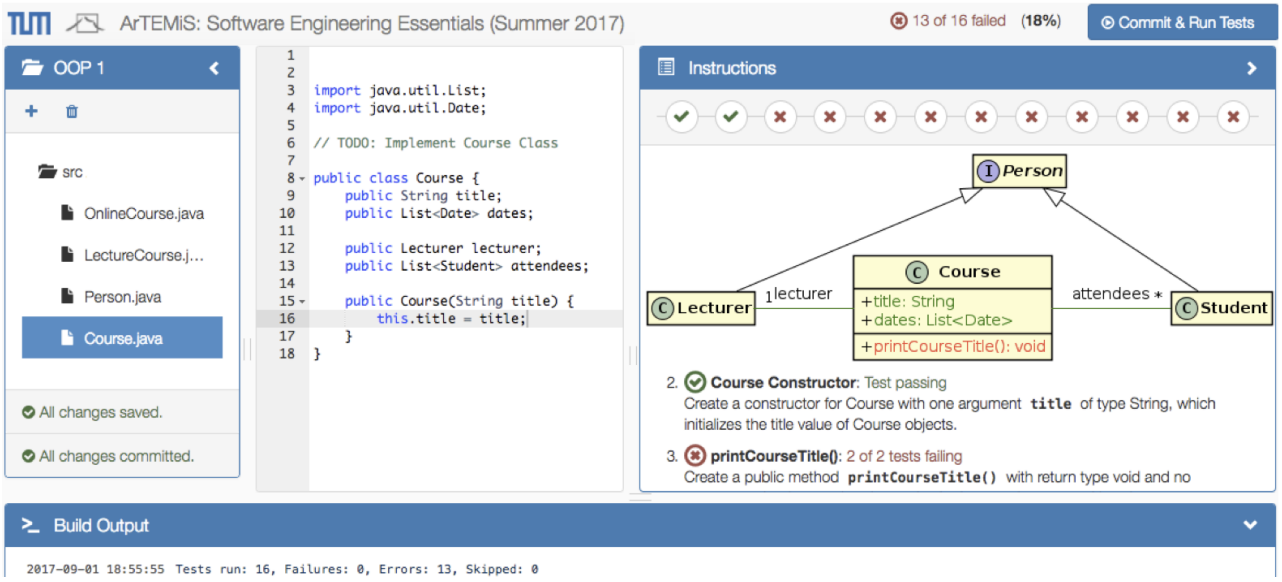

Figure 3: Screenshot of the ArTEMiS online editor with interactive exercise instructions on the right

test. The student can now reattempt to solve the exercise and submit a new solution. The instructor can review the results, gain insights on the exercise progress and react immediately to errors and problems during the exercise. ArTEMiS acts as facade to the CI process and hides complex details, which enables less experienced students to participate in the exercise. Every student has a personalized build plan, so the approach can be used to teach the concepts of CI. Then, students get access to their build plan and have to configure it on their own. Students can only see, edit and adjust their personalized build plan. They cannot inspect build plans of their fellow students.

ArTEMiS includes an online editor that allows unexperienced students to participate in exercises without dealing with the complex setup of VC and IDEs. Figure 3 shows the online editor with interactive and dynamic exercise instructions on the right side. Interactive instructions change their color depending on the progress of students. Already completed tasks are marked with a green tick, incomplete tasks are marked with a red cross. This helps students to identify which parts of the exercise they have already solved correctly. When they submit their current solution with the *Commit & Run Tests* button in the upper right corner, the interactive instructions dynamically update. The exercise tasks and the UML diagram elements are referenced by the predefined test cases. They change their color from red to green when all test cases associated with the task or diagram element pass. This allows the students to immediately recognize which tasks are already fulfilled and is particularly helpful for beginners.

Figure 4 shows the system architecture of ArTEMiS. A student uses the *ArTEMiS Application Client* (a browser) and a *VC Client* of his choice to obtain the exercise code and to submit solutions. *VC Server*, *CI Server*, *Local Build Agents*, and the *ArTEMiS Application Server* run on the university's infrastructure. The *CI Server* delegates the builds to local and/or remote agents, e.g. on Amazon Web Services, depending on how much capacity is required for the number of participating students. This makes it easy to scale the approach by adding additional build agents. ArTEMiS uses the university's *User Management System*. The components *VC Server* and *CI Server* are exchangeable, resulting in a flexible system which can be adapted to the specific requirements of instructors.

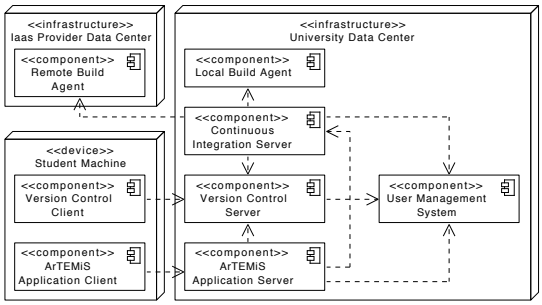

Figure 4: System Architecture of ArTEMiS

The use of CI leads to several benefits: the programming language for programming exercises is freely configurable. Compilation and testing of student solutions is not a matter of ArTEMiS, it only depends on the configuration of the build system. Students learn the concepts and workflows of VC, CI and testing, all important skills in software development. The system is scalable and can react to the number of assessments to be handled with a corresponding number of build agents. Adding more build agents allows more students to submit their solution at the same time and still receiving instant feedback. This is crucial for interactive learning based exercises that are integrated into a class.

4 CASE STUDY

We use ArTEMiS in 3 large university course (UC) and in a MOOC:

(1) **UC - Introduction to Software Engineering**: mandatory subject, 1400 bachelor students (2nd semester), 6 exercises, 814 students participated
(2) **UC - Patterns in Software Engineering**: elective subject, 400 master students, 34 exercises, 334 students participated

(3) **UC - Project Organization and Management**: mandatory subject for business informatics, elective subject for computer science, about 300 bachelor and master students, 1 exercise, 224 students participated
(4) **MOOC - Software Engineering Essentials**: 300 active students, 10 exercises, 257 students participated

In the beginning of these courses, we go through a short interactive tutorial together with the students to show them how to use ArTEMiS. In all these courses, we recommend to use Eclipse as IDE and SourceTree as VC client (git). Students could individually decide which VC client to use to retrieve the assignment, and hand in their solution. We describe 4 typical exercises from our courses:

(1) **Programming:** write source code for a problem statement or UML model. Typical examples are the implementation of the quick sort algorithm or the strategy pattern. Test cases assess the correctness of the solution.
(2) **Testing:** write test cases for the given program code and mock parts of the program. The instructor creates test cases that assess the test cases of the students with the given correct program code and by injecting wrong program code to review if the tests of the students are correct.
(3) **Merge conflict:** experience a merge conflict in git and have to resolve it.
(4) **Release management:** learn how commits trigger the CI server, which produces new build artifacts. Students get direct access to their personalized build plan to be able to view its configuration in detail.

By automizing all setup steps, we decreased the effort for students and instructors. The instructor can dynamically adapt the working time for different assignments based on the students' progress in class. This allowed students to actively think about the exercises instead of just following along with the solution.

4.1 Results

In simulations with 800 students and 2000 submissions over a period of 10 minutes, ArTEMiS assessed solutions in an average of 10 seconds. In our courses, we measure the average assessment time. This allows us to constantly evaluate whether ArTEMiS fulfills its scalability goals under real classroom conditions. In a quantitative analysis, we found the following results:

(1) Scalability: ArTEMiS can handle 200 submissions per minute.
(2) Feedback: ArTEMiS provides feedback within 10 seconds.
(3) Usability: Programming beginners are able to use ArTEMiS.

Table 1 displays an overview of the number of students and submissions for each exercise together with the average assessment time. The number of submissions per student varies from 1.6 to 4.3 on average. In one exercise, where students could only use the online editor, this number was higher: 7.2. If students participate in the exercise and submit solutions, most of them also successfully solve the exercise. The assessment time, i.e. the time for students to wait from the moment they submit their solution until they see the test results and the feedback, varies from 5.1 to 10.3 seconds on average depending on the complexity of the exercise, the number of tests and the number of external dependencies. In exercises with more complex tests, e.g. asynchronous client-server tests with timeouts, this number can increase. In such cases, it makes sense to distribute tests to students so that they can also execute them on their computer. ArTEMiS uses a separate test repository so that students who try to cheat and change tests locally, e.g. to let all tests pass immediately without solving the exercise, can be detected.

We asked the students for feedback regarding the use of ArTEMiS. Most students had less experience in the areas of distributed VC and CI, nonetheless they had no issues working with ArTEMiS. They stated, that the test results and the feedback was helpful to solve the exercises, they enjoyed working with it and preferred the usage of ArTEMiS over the previous process where there was no automatic and instant feedback. In an online questionnaire, we found that more than 90 % of the students consider the interactive exercise instructions helpful in solving the exercise. They are particularly valuable in online courses, where students are distributed and instructors cannot guide them directly in case of problems.

	(1) Programming	(2) Testing	(3) Merge conflict	(4) Release management
Participating students	317	167	224	248
Submitting students	209 (66 %)	109 (65 %)	211 (94 %)	149 (60 %)
Successful students	200 (96 %)	108 (99 %)	183 (87 %)	135 (91 %)
Overall submissions	340	340	904	285
Correct submissions	236	236	291	198
Test cases	12	12	2	0
Assessment time*	10.3 s	8.3 s	5.1 s	9.6 s
Submissions per student*	1.6	3.1	4.3	1.9

Table 1: Numbers in typical exercises (* on average)

4.2 Discussion

ArTEMiS provides flexibility in how instructors conduct exercises. It allows them to distribute assessment tests to students so that they can find their own errors during debugging easier. However, this might facilitate that students only work on getting the tests passed and do not take the time to understand the actual problem and solve it on their own. In such cases, instructors can hide the tests and only show the test results on ArTEMiS.

Another choice is the use of the online editor vs. the use of an IDE on the local computer. While the online editor lowers the entrance barrier, it has limited features. In a comparison, we found that students prefer the local IDE if they are already familiar with it, as it offers more features such as syntax and error highlighting, auto completion and debugging. In an experiment, we forced them to use the online editor, however some students copied the code file by file into their local IDE, solved the exercise there, and copied the code back. An open question at this point is whether providing features such as auto completion is beneficial to the learning experience. Novice programmers often heavily depend on such features [18]. This may make students reliant on them and may prevent learning the correct syntax of the programming language. It could be a viable strategy to provide only minimal features in a code editor.

ArTEMiS supports in-class exercises with hundreds of students and can additionally be used for homework exercises. We use it in online courses where students are distributed and rely on the exercise instructions and in university classes where instructors can guide the students in addition. The costs of providing the system are not negligible. We could use already existing, self-hosted CI and VC systems. These are hosted at our institution, but we have to take care of the maintenance. Alternatively, cloud-based solutions such as GitHub Education, GitLab or Bitbucket Cloud can be used. These usually offer attractive opportunities for educational institutions.

5 RELATED WORK

A variety of systems for automatic assessment exists. Multiple surveys have been published, summarizing and categorizing these systems. The first extensive surveys on this topic were done by Ala-Mutka [1] and Douce et al. [5]. They describe multiple auto assessment tools and categorize them into dynamic and static assessment and differentiate between local and web based systems.

The survey by Ihantola et al. focuses on identifying key features of automatic assessment tools, such as supporting different programming languages, allowing resubmissions, or providing a sandbox environment to handle malicious submissions [10]. The authors state most systems are not open source or available otherwise, even if a publication describes the development of a prototype. A survey by Queiros states interoperability and compatibility to other services is a key factor for automatic assessment systems [14]. He concludes that this factor is not considered for many existing assessment tools and that future solutions have to improve this. Our approach fulfills many of the stated features by connecting to university user management or providing interfaces to VCS.

From those surveys, we identified multiple publications related to our system. WebCAT was first created in 2003 and is arguably one of the most complete automatic assessment tools [6]. It has been developed as open-source software and allows extensibility by plugins. In terms of assessment, it supports student written tests, test coverage, static code analysis and a combination of both automatic and manual grading. Our approach covers most of the features of WebCAT, while removing the dependence on a single software product. Instead, our approach consists of multiple independent software systems that are connected using common interfaces. This leads to a higher flexibility as individual parts of the architecture can be replaced, for example in favor of lower costs, superior support, larger communities or general management decisions.

Marmoset focuses on information collection during the development process of students [16]. The system takes regular snapshots of the students' progress. It allows the instructor to study the development process of the students and to identify common bug patterns. By using VCS and teaching its application, we achieve the same outcome. Students commit multiple iterations of their solution, resulting in a commit history that can be evaluated. This allows to identify common mistakes and study problem solving behavior. Amelung et al. propose a system thats splits e-learning and e-assessment platforms into separate systems, allowing independent deployment and easier adoption [2]. Our approach targets the same idea, but our goal is not to implement another closed source assessment system. Instead, we reuse workflows provided by existing CI tools to achieve similar results.

Gruenewald et al. focus on the challenge of conducting programming lectures as MOOCs [8] integrating active experimentation and relating to concrete experience. Those aspects are considered in ArTEMiS as well. In recent years, commercial products have become available for automatic grading, including Vocareum, Turing Craft, etc. An open source and free alternative is Codeboard, developed by ETH Zurich. These online tools are cloud based and use regression testing, a technique used for quality control in software development. They allow students to edit and submit source code in the browser to simplify the participation.

6 CONCLUSION

ArTEMiS combines VC and CI with automated assessment of programming exercises and immediate feedback. This enables high flexibility and scalability in large classes. Our experiences in 3 university courses and 1 online course show that programming beginners are able to use the system, improve their solutions iteratively with immediate feedback and increase their learning experience.

Dynamic and interactive exercise instructions are particularly helpful for beginners to immediately recognize which tasks are resolved. The effort for instructors and TAs is reduced. They can evaluate student results immediately during the exercise to help students when problems occur. ArTEMiS is free and open source on https://github.com/ls1intum/ArTEMiS, so that other instructors can use it in their courses. We will support additional interactive exercises in the future, in particular quizzes and modeling.

ACKNOWLEDGMENTS

We want to thank Dominik Münch, Andreas Greimel and Josias Montag who participated in the development of ArTEMiS.

REFERENCES

[1] K. Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* (2005), 83–102.
[2] M Amelung, P. Forbrig, and D. Rösner. 2008. Towards Generic and Flexible Web Services for E-Assessment. *SIGCSE Bulletin* (June 2008), 219–224.
[3] C. Bonwell and J. Eison. 1991. *Active Learning: Creating Excitement in the Classroom.* ASHE-ERIC Higher Education Reports.
[4] G. Booch. 1991. *Object Oriented Design with Applications.* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.
[5] C. Douce, D. Livingstone, and J. Orwell. 2005. Automatic Test-Based Assessment of Programming: A Review. *Journal on Educ. Resources in Computing* (2005).
[6] S. Edwards. 2003. Improving student performance by evaluating how well Students test their own programs. *Journal on Educ. Resources in Computing* (2003).
[7] M. Fowler. 2006. Continuous Integration. http://www.martinfowler.com/articles/continuousIntegration.html. (2006).
[8] F. Grünewald, C. Meinel, M. Totschnig, and C. Willems. 2013. Designing MOOCs for the Support of Multiple Learning Styles. In *European Conference on Technology Enhanced Learning*. Springer, 371–382.
[9] J. Hollingsworth. 1960. Automatic Graders for Programming Classes. *Commun. ACM* (1960), 528–529.
[10] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Koli Calling Conference on Computing Education Research.* ACM, 86–93.
[11] M. Joy, N. Griffiths, and R. Boyatt. 2005. The BOSS Online Submission and Assessment System. *Journal on Educational Resources in Computing* (2005).
[12] S. Krusche, A. Seitz, J. Börstler, and B. Bruegge. 2017. Interactive Learning: Increasing Student Participation through Shorter Exercise Cycles. In *Proceedings of the 19th Australasian Computing Education Conference.* ACM, 17–26.
[13] S. Krusche, N. von Frankenberg, and S. Afifi. 2017. Experiences of a Software Engineering Course based on Interactive Learning. In *Proceedings of the 19th Workshop on Software Engineering Education in Universities.* 32–40.
[14] R. Queirós and J. Leal. 2012. Programming Exercises Evaluation Systems – An Interoperability Survey. In *Conference on Computer Supported Education.* 83–90.
[15] R. Singh, S. Gulwani, and A. Solar-Lezama. 2013. Automated feedback generation for introductory programming assignments. *SIGPLAN Notices* (2013), 15–26.
[16] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, . Hollingsworth, and N. Padua-Perez. 2006. Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses. *SIGCSE Bulletin* (2006), 13–17.
[17] T. Staubitz et al. 2015. Towards Practical Programming Exercises and Automated Assessment in Massive Open Online Courses. In *International Conference on Teaching, Assessment, and Learning for Engineering.* 23–30.
[18] A. Vihavainen, J. Helminen, and P. Ihantola. 2014. How novices tackle their first lines of code in an IDE: Analysis of programming session traces. In *Koli Calling Conference on Computing Education Research.* ACM, 109–116.
[19] M. Vujošević-Janičić, M. Nikolić, D. Tošić, and V. Kuncak. 2013. Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments. *Inf. Softw. Technol.* 55, 6 (2013), 1004–1016.