HAVING IT ALL: AUTO-GRADERS REDUCE WORKLOAD YET INCREASE THE QUANTITY AND QUALITY OF FEEDBACK

M. Nurminen [*]
Tampere University
Tampere, Finland
ORCID 0000-0001-7609-8348
P. Niemelä
Tampere University
Tampere, Finland
ORCID 0000-0002-8673-9089
H.-M. Järvinen
Tampere University
Tampere, Finland
ORCID 0000-0003-0047-2051

Conference Key Areas: Open and Online Teaching and Learning, Integrated learning environments for the digital native learners

Keywords: Online assessment, automatic grading, online course, web development

ABSTRACT

Due to COVID-19, teaching has moved online at an accelerated pace, and this movement will partially be permanent. Online teaching implies an automatic assessment of exercises. Using automated grading, the studied web development course (N=257) managed to serve students promptly and increase the amount of feedback received by students even if the number of submissions increased remarkably.

Automatic graders guaranteed the uniformity of feedback, equal treatment, and most importantly, reduced the routine work of the personnel. Being less burdened, the course personnel could concentrate on assisting students in online discussion channels, where discussions were targeted for the students needing more help and support. Compared with previous manually assisted course implementations, the workload moved from ”in situ” to prior to the course, where the most laborious part was the design of the exercises and the implementation of automatic graders. The amount of work for grading the exercises and assignment was decreased by about 70 per cent.

In the graders, the feedback given by them is of paramount importance and should suggest necessary improvements. The graders enforced good coding conventions and other targets set for the code (e.g., maintainability and accessibility). In some cases, this feedback was modified during the course based on the difficulties experienced to give more targeted advice. Automatic grading provided a way for students to iteratively improve their code based on the feedback. The software and methods used in this course could be applied to such other courses and domains, where automatic grading is considered helpful.

[*]Corresponding author
M. Nurminen
mikko.nurminen@tuni.fi

1 INTRODUCTION

The International Association of Universities (IAU) studied the impacts of COVID-19 on higher education institutions (HEIs) and reports about remarkable changes worldwide, e.g.: 59% of HEIs have ceased all on-campus teaching, and two-thirds of them reported replacing classroom teaching with online distance teaching. While acknowledging the difficulties of these transitions, the study findings highlight the opportunities afforded by “more flexible learning possibilities, exploring blended and hybrid learning, and mixing synchronous learning with asynchronous learning.” [1] The effects of COVID-19 may not be only temporary, but are anticipated to alter educational practices permanently.

Even before the pandemic, the trend had been in the direction of online education and larger student groups. For example, in 2019, the Finnish Ministry of Education set a goal of utilising digital environments and artificial intelligence in learning on a larger scale [2]. In the studied Web programming course, hereafter WebDev1, the goal legitimized the effort to make all the exercises automatically graded. In the previous course implementation, the grading of weekly exercises was already automatized. In this implementation, the effort was made to automate the grading of the assignment and exam as well.

In online learning, students often need to be more autonomous, as scaffolding from the course personnel is only available in online forums. Lecture slides and links to internet materials were provided. Also short hands-on and lecture videos were created – short, because cutting the material in shorter portions has proven to increase student engagement in the earlier studies [3–5]. While adherents of ‘flipped learning’ promote shortening videos with one voice, other details are still under debate, such as, which length then is the most optimal (according to Bergmann five minutes [4]), and whether the same size fits all (males and students with learning disabilities tend to favor shorter videos [5]).

Programming exercises need to be designed and instructed so that a student can do them with as little additional support from the course personnel as possible. In addition to well-structured exercise instructions, exercise graders should provide sufficient feedback for students. In order to cover most potential error sources in student’s code and to give proper, actionable feedback for fixing them, grader’s code may be many-fold compared with the student’s submission. Often the code for an exercise’s grader would have several hundreds of lines of code, whereas complete student submission to the exercise could be well under a hundred.

In this article, we compare the time required by automatic vs. manual grading, and discuss the effect the automatic grading had on students’ code quality. We also compare the course’s processes and tools to those used in the software industry, as a way of validating these choices.

2 RELATED WORK

Peer reviews and automatic grading are two of the methods which have been studied for lessening the personnel’s workload on courses with scarce resources. In an earlier implementation of the WebDev1, automatic grading was used in the first half of the course and peer-reviews in the latter. In the course’s post-survey, students’ attitudes towards automatic grading were more positive than for peer-reviews. [6]

Obviously, manual grading requires more teaching resources if the number of students increases [7]. However, teaching resources are not that easily available, and if inexperienced TAs are extensively used, the quality of feedback and the variety in given points start to increase, leading to unequal treatment of students. In their study, Leite et al. claim that students who received human feedback perform slightly better than those who receive automatic feedback [8]. Quiz and exam results, and course grades showed human feedback led to better conceptual understanding and better performance overall. As a result, the study deduces that human-provided feedback about the relation of the syntax and logic in students’ code could be a primary mechanism for human feedback to improve learning outcomes.

Software quality is a widely studied field, Boehm et al. formed the Software Quality Characteristics Tree from the related terms [9]. Related to software quality, the feedback given to the student’s code submissions can affect their learning negatively or positively, positively if it helps them on their path to good performance and better code quality.[10] Feedback in the teaching of programming has been studied earlier, for example Stegeman et al. suggested a rubric for feedback [11], as do Marceau et al. when studying the effects error messages had on learning[12]. The effect of the feedback on the software quality in the context of a university programming course. There is a tension in teaching programming in university: how much of the teaching should concentrate on the pure theory, and how much time should be given to teaching practical programming skills.[13], usually students prefer to rapidly learn coding skills which lead to employment. This tension can partly be eased by selecting tools and processes for the course which are already in use in the industry.

Plussa development has been paralleled with a study of different exercise and assessment methods and their pedagogical value. In addition to auto-grading, various learning activities such as visualizations of different algorithms [14, 15] and runtime behaviour [16] have been on focus. Runtime visualizations illustrate, e.g., call stack and heap behavior while executing code (e.g., Annotation editor exercise about recursion). Visualizations are an apt tool for lowering the threshold of difficult topics, and, e.g., WebDev1 exploited Loupe - event loop visualization in internalizing the JavaScript concurrency model. However, visualization systems are often short-lived research prototypes where the user controls the program animations [17]. Yet these comprehension aids are good for novices, but more advanced students, such as in WebDev1, do not need toys but real tools for gaining experience. The writers demonstrate the utility of GitLab as a dissemination and grading tool in integration with Plussa learning management system [18]. Since GitLab also provides some DevOps capabilities, WebDev1 aimed at acquainting students with these DevOps practises, i.e., to teach GitOps on the side. GitOps could be further extended with Kubernetes, which would provide a fully-fledged automated orchestration solution for the courses of Web&Cloud domain [19].

Inspired by the earlier studies, we set the following RQs:

1. How do the TAs workload and the intensity of work differ in automatically graded courses when compared to those that are manually graded?
2. How does auto-grading affect the quality of code?
3. Are the course's processes and tools similar to ones used in the industry?

3 RESEARCH CONTEXT

WebDev1 provided a comprehensive introduction to both front-end and back-end web technologies: front-end technologies consist of HTML5, CSS, and JavaScript, whereas the back-end introduces Node.js. Unlike previous years, the utilization of Node.js frameworks, such as Express and Handlebars, was omitted. Instead, the vanilla JavaScript approach was used primarily for pedagogical reasons: frameworks come and go, but HTTP and generic client-server architecture will stay. The course is targeted to third- and fourth-year students. The prerequisites for this course include three basic programming courses, and a basic database course. Prerequisites imply that course participants should have a considerable amount of programming routine, including a basic understanding of project work, e.g., Agile project management.

The WebDev1 course will be developed in iteration cycles twice a year. The development started in the 2019-2020 academic year [6]. In the next academic year, it was continued by the introduction of auto-graders for assignment containing both unit tests but substantially more static code analysis. Cyclic development with reflective redesign phases is characteristic of design-based research (DBR) [20] [21] [22]. DBR mandates a guiding background theory, and this study leans on the previous findings of flipped learning in the course arrangements [23–25]. In DBR, educational solutions are combined with the empirical interventions and proof: DBR systematizes course development cycles of *design, development, enactment, and analysis* [26–28]. Here the *cycle* represents a course term. The retrospective analysis inserts requirements into the design of the next implementation [29–31]. The redesign implies *'reflective conversation with the situation'* [32], whereby course personnel observe the effects of new arrangements and refine them if necessary.

The study was conducted during the global COVID restrictions, where moving to remote teaching was a general recommendation. Thus, WebDev1 course replaced previous lectures with video recordings and on-premises tutoring with online tutoring sessions. Students struggling with the exercises or the coursework assignment could get help during these so-called Kooditorio sessions, which were held in Teams. Kooditorio is a tutoring practice a-kin to primetime [33], except voluntary, where teachers and assistants answer questions, debug and co-implement students' code and scaffold them finalizing their exercises.

3.1 Tools used: Plussa and Gitlab

Learning management system Plussa was used during the course [34] to host course materials such as slides, exercises and videos. The videos handled the subject matters of the week, and were largely based on the lecture slides. A few selected topics were introduced by visiting lectures, such as accessibility and security. For some weeks there were also hands-on videos, which demonstrated using specific technologies. Personnel were inspired by the principles of flipped learning, where short videos and related exercises take turns.

In addition to Plussa, Gitlab is a central tool throughout the course. Gitlab functions as a normal version control system, but also provides means for project management and DevOps. The course personnel create students' and groups' Git projects to Gitlab using an in-house tool named Repolainen. Plussa submissions are done by giving the GitLab URL of one's repository. Repolainen is also in charge of communication with other systems, such as Gitlab, or SonarQube static code analysis.

The creation of the student repositories is done at the beginning of the course, group repositories are created after group formation. To create them, Repolainen is fed a list of students, or group's members. Course personnel are given maintainer level permissions, students are granted developer permissions. CI pipeline was introduced to the students, as they will go deeper into DevOps in their further studies. Gitlab CI pipelines are configurable with the .gitlab-ci.yml file. This file could be edited by the student groups in their own repository. Exercise instructions were either in Plussa or in the Git upstream repository, sometimes in both. The 'course upstream' is a Gitlab repository for pulling only. Course personnel maintain the upstream, new instructions and possible file skeletons are released at the beginning of each exercise round.

The assignment started with the creation of GitLab group repositories. Gitlab Issue Board was recommended as a tool for project management to coordinate tasks. The Issue Board provides a Kanban-like issue management view, where issues can be moved in steps from the backlog, to the 'Doing' and finally to the 'Closed' board. These moves inform other group members not to touch on-going work. A couple of Plussa exercises were used to orient students in using the Issue Board. In the assignment instructions, the required documentation included an appropriate use of issues: groups were advised to list user stories as issues, and assign tasks in the Gitlab Issue Board. All in all, when correctly applied, issues provided a panoptic view of the progress of each group.

3.2 Automatic grading

To complete the course, students had to pass weekly exercises, a coursework assignment, and an exam. The maximum course grade was five: +1 for weekly exercises, +2 for assignment, +2 for the exam.

The grading of exercises and the coursework assignment was automated where possible. Without automation, the amount of work would have been enormous, the theoretical maximum total number of submissions was 205.600. Course personnel of three could not have assessed this number of submissions manually. Maximum number of submissions $N_{subs.}$ can be calculated using equation 1:

$$\begin{aligned} N_{subs.} &= N_{students} \times N_{modules} \times N_{exercises/module} \times N_{subs./exercise} \\ &= 257 \times 10 \times 8 \times 10 = 205.600 \end{aligned} \tag{1}$$

The level of automation has increased remarkably during successive course implementations: in 2019 half of the course was auto-graded [6]; in 2020 everything but documentation and 'UI wow' were auto-graded.

For the coursework assignment students were paired, which resulted in 257/2 = 129 groups. In 2020, the groups implemented on-line shops. Exercise rounds eight through ten comprised the mandatory part of the assignment: having passed the tenth exercise round students received a passing grade for the coursework assignment. Then students chose either to accept this result or to continue to higher grades. This can be interpreted as a partial application of flipped assessment [35]: students can 'select' the grade they are after. Level1 implied a grade one for the assignment. The level 1 assessment was fully automated including Mocha tests and JSDoc linting. Level2, in turn, implied a grade of two, and also contained parts left for course personnel to assess manually, such as the quality of documentation and the usability and prestige of UI, the so-called 'UI wow'. Level2 cumulatively adds more automatic tests to Level1, with automatic graders for functional programming, eslinting, static code analysis with SonarQube, and coverage.

Fig.2 illustrates the process of auto-grading. Process starts when a student commits code to Gitlab, and then submits their Gitlab URL on an exercise page in Plussa. As a system, Plussa divides into two parts, both run as Docker containers: Plussa frontend "run-aplus-front container in the picture", and MOOC grader, "run-mooc-grader". Plussa frontend provides the UI, and maintains a grade repository.

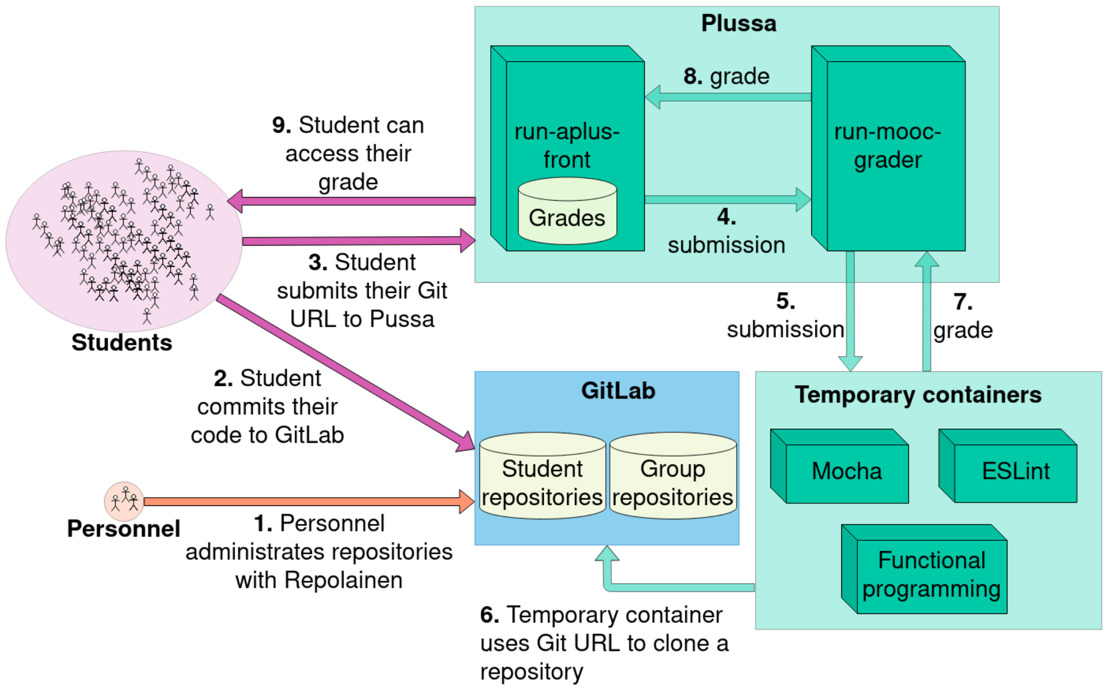

Plussa
run-aplus-front
Grades
8. grade
run-mooc-grader
9. Student can access their grade
4. submission
3. Student submits their Git URL to Pussa
Students
5. submission
7. grade
2. Student commits their code to GitLab
GitLab
Student repositories
Group repositories
Temporary containers
Mocha
ESLint
Functional programming
Personnel
1. Personnel administrates repositories with Repolainen
6. Temporary container uses Git URL to clone a repository

Figure 1: The interplay of Plussa, GitLab and Repolainen in auto-grading

MOOC grader, in turn, provides the exercises and takes care of grading. It launches temporary Docker containers that are started only to perform the grading. In Fig.2, ESLint, functional programming, or Mocha graders are examples of such graders. Usually, the grader clones a student's git repository and executes the grading as instructed in a shell script.

Since testing was not particularly central in the course curriculum, most of the tests were given to students purpose-built to familiarize with Mocha and its execution; respective Plussa graders ran the same tests. In local tests, students received the same feedback as given by the Plussa graders, which decreased the number of needed submissions. Running the tests locally gave students a view of how their work would be graded in Plussa.

3.3 Method and research instruments

We looked at how the feedback from the automatic graders affected the quality of students' code, as evident from the number and type of errors reported by the automatic graders. The tools and processes selected for the course were evaluated by comparing them with those reported in StackOverflow's Developer Survey (SODS)[36] with about 65,000 responses from software developers from 186 countries, and JetBrains' The State of Developer Ecosystem (JBSODE) survey of 19,696 developers[37].

4 RESULTS AND DISCUSSION

4.1 Comparison of work workload and intensity between auto- vs manually assessed course

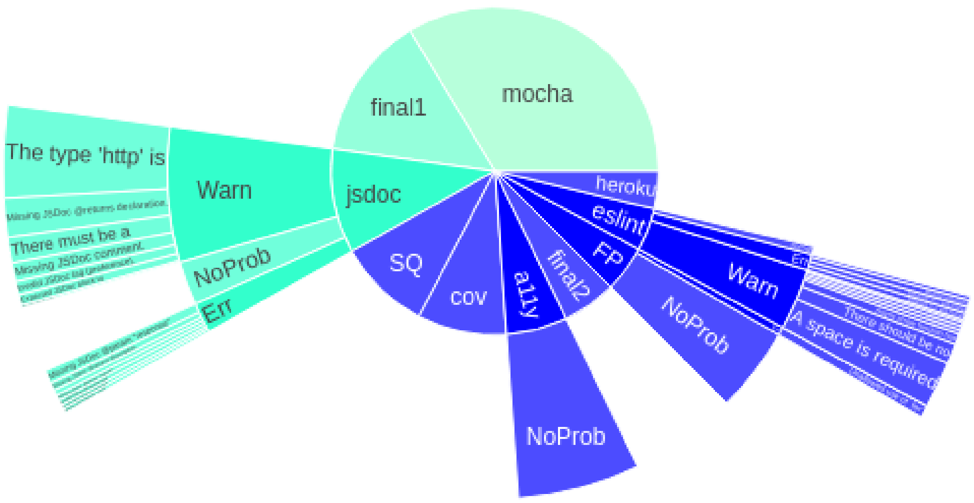
The answer to this RQ can be estimated based on the current automatically-graded WebDev1 course implementation, and previous manually graded basic web programming courses. On the current implementation, practically everything was automated, including the grading of the exercises, group assignment, and the exam in Plussa. Personnel worked on the design and implementation of these. The current course had 50 automatically graded exercises with graders, and 9 graders for the assignment. When an estimated 12 hours on average was spent on the design and implementation of a grader, the total hours were 59 * 12h = 708h. This course implementation featured newly designed graders, and in future implementations these can be used as the basis for creating others, thus reducing the required time.

Based on similar earlier courses in Tampere University, when manually grading and giving feedback a TA could use an estimated 15 minutes per exercise, and 1 hour in grading assignment. In manual grading, the number of students becomes significant: the course's assignment stage was participated in by 173 students in 85 groups. For the exercises the required TA work time for grading and feedback would equal 0.25h * 50 exercises * 173 students = 2165,2h. Grading the group assignment would take 1h * 85 groups = 85h. The combined time consumed is 2247,5h. Thus, the time required for manual grading is far greater than for creating the automatic graders (708h). It is resource-wise a sound decision to automatically grade as many exercises as possible.

4.2 How does auto-grading affect the quality of code?

The assignment complied with the principles of flipped assessment [35], where students may select a harder assignment if they estimate themselves to be competent enough. Students could choose between No assignment, Level-1 or -2. Fig. 2 illustrates the graders color-coded into Level-1 (cyan) and Level-2 (blue) graders. Each passed level improves grade with +1. Level-1 was tried by 126 students (for reference, 157 concluded the course). By far the most frequent was mocha unit test grader. ‘No pass’ in mocha led to giving up the assignment and filtered submitters for later jsdoc and final1 graders. Final graders ‘final1’ and ‘final2’ combined separate graders of respective levels and executed all tests in a sequence. This prevented the manipulation of a submission: e.g., ‘pass’ could be ensured only with a selected functionality, and after the pass, the quality of code could again be compromised.

After the final1 grader, there was no use to continue without passing Level1, thus, in the transfer to Level-2, the number of students decreased by half. Most of the Level-2 submitters were testing with eslint first. Besides being the first in the list, eslint or linting in general is utilized in other courses as well, so many students are familiar with it and its functionality as a grader is straight-forward. Gradual improvement is evident based on the students’ submissions. The first submission was often very buggy, almost like going on a fishing expedition. Once students got a grasp of what is the spirit of a game and how exactly the grading is done, the errors converged to zero quite rapidly. Characteristic of the error hunt was a non-stop process, where subsequent submissions followed each other at high frequency.



px

Figure 2: Auto-grading and provided error and warnings

Table 1 illustrates the most common error and warning types in a descending order of occurrences. Here, the most common eslint errors, such as strict comparison and semicolon issues could be focused on more thoroughly during the lessons based on the analysis. In addition, warnings expose defects with async-await in combination with arrow functions.

4.3 Are the course’s processes and tools similar to ones used in the industry?

In both SODS (69.7%) and JBSODE (70%), JavaScript was the most commonly used programming language. On the SODS list of most liked programming languages, JavaScript is in the tenth position in the rankings of most used languages, TypeScript which builds on JavaScript was in second place. In the rankings of languages developers would like to work with, JavaScript was in second place, TypeScript being fourth. So, the language selection of the course gave students experience with languages they will likely use in the future, as current developer preferences can determine the languages selected for upcoming projects. In SODS the category of ‘Other frameworks, libraries, and tools’, Node.js is ranked as the most desired future tool. MongoDB fares well in the category of ‘Databases’ coming third in ‘the most used’ ranking.

Table 1: The occurrences of errors and warnings in a detail

grader	cat	type	val
mocha			765
jsdoc	NoPrb		49
	Err	Missing JSDoc @param response description.	10
	Err	Missing JSDoc @returns description.	6
	Err	Missing JSDoc @param request description.	4
	Err	Invalid JSDoc @returns type Object; prefer: object.	4
	Err	Missing JSDoc @param userId description.	3
	Err	Invalid JSDoc @param currentUser type Object; prefer: obj	3
	Err	Invalid JSDoc @param userData type Object; prefer: object	1
	Err	Missing JSDoc @param password description.	1
	Err	Missing JSDoc @param filePath description.	1
	Err	Invalid JSDoc @param user type Object; prefer: object.	1
	Warn	The type ’http’ is undefined.	69
	Warn	Missing JSDoc @returns declaration.	29
	Warn	There must be a newline after the description of the JSD	19
	Warn	Missing JSDoc comment.	14
	Warn	Invalid JSDoc tag (preference). Replace return JSDoc ta	10
	Warn	Expected JSDoc block to be aligned.	8
	Warn	Missing JSDoc @param response type.	2
final1			322
heroku			84
eslint	NoPrb		9
	Err	Expected ’===’ and instead saw ’==’.	5
	Err	Missing semicolon.	5
	Err	Expected ’!==’ and instead saw ’!=’.	5
	Err	Global variable leak, declare the variable if it is inte..	1
	Err	Identifier ’current$_{u}ser'isnotincamelcase.$	1
	Err	Unexpected var, use let or const instead.	1
	Warn	A space is required after ’,’.	35
	Warn	There should be no space before ’,’.	15
	Warn	Async arrow function has no ’await’ expression.	8
	Warn	Missing space after =>.	3
	Warn	Use an object spread instead of ‘Object.assign‘ eg:	3
	Warn	Missing space before =>.	3
	Warn	Async function has no ’await’ expression.	2
cov			200
FP	NoPrb		95
	Err	Unallowed use of ‘for‘ loop	9
a11y	NoPrb		137
SQ			209
final2			121

Git-related collaboration tools were ranked high in ‘Collaboration tools’ in SODS, with GitHub being top-ranked and GitLab fifth for ‘professional developer’ respondents. During WebDev1 students used Git and GitLab extensively, so they gathered valuable experience with version control and issue management. Further, WebDev1 encouraged students to experiment with DevOps by creating and assigning issues and running a CI pipeline. In SODS half of the respondents see DevOps as ‘extremely important’, and that their organization has at least one person working on DevOps, while in JBSODE half of the respondents were involved in DevOps to some extent. While WebDev1’s DevOps treatment was quite light, students acquired knowledge and experience with the basics of DevOps.

5 CONCLUSIONS

In WebDev1, auto-grading decreased the effort spent with routine tasks by 70%, yet the amount of feedback, the consistency of it, and submissions made by students all increased. By examining the submissions, the improvement of code quality was obvious: most students kept iterating till they managed to pass both the functional tests and static analysis of the code. The pass was rewarded with a better grade that being allegedly the major motivation. However, compared with “black-box assessment” done by the personnel (or a peer), incremental improvement of code, where students are in control of the process, can be seen as the source of empowerment. In addition, the grading system used complies with the DevOps practices of industry, therefore training students better for their future and increasing their employability. Having it all done – faster and better than expected – the course personnel can rejoice all their way till well-deserved summer holidays.

6 FURTHER STUDIES

Data collected by Plussa and GitLab is massive and would provide material for learning analytics; the results should be accessible for both teachers and students. Students could be keen on performance comparisons, though this might induce unnecessary competition. Comparing students’ performance to their own earlier performance is safer. Current Plussa graders check code quality and conventions. In addition, a grader visualizing the learning process would be handy in improving students’ consciousness of their strengths and weaknesses, preferably with suggestions of exercises to fill the gaps. The anticipated grader is called a self-reflection grader.

Another interesting research path would be investigating the most pedagogically fruitful way of combining automatic grading and the teaching and support provided by the course personnel. While automatic grading was shown to be effective and also sensible resource-wise, during course implementations numerous students have expressed their need for support from the course personnel, and time saved with automatic grading could enable giving this support. Here Teams channels were useful in student-peer and student-teacher interactions. But especially during the current COVID pandemic, which places more psychological strain on students, design-based research course design process should integrate student support to the design phase with instructional design. As an example, one aspect of the course this integration could improve is teacher-student communication. Currently the interaction strategy on the course is focused on selecting the appropriate tools, such as Teams channels or emails. How these tools are used: what is communicated, using which tool, and by whom is often decided in ad hoc manner. A more structured approached would make communication during implementations more predictable.

References

[1] Giorgio Marinoni, Hilligje Vant Land, and Trine Jensen. The impact of covid-19 on higher education around the world. *IAU Global Survey Report*, 2020.

[2] Korkeakoulutuksen visio 2030, 2019. URL `https://minedu.fi/korkeakoulutuksen-ja-tutkimuksen-visio-2030`.

[3] Fiona Saunders, Sandor Gellen, Jack Stannard, Colin McAllister-Gibson, Lisa Simmons, and Andy Gibson. Educating the netflix generation: Evaluating the impact of teaching videos across a science and engineering faculty. 2020.

[4] Jonathan Bergmann and Aaron Sams. *Flip your classroom: Reach every student in every class every day*. International Society for Technology in Education, 2012.

[5] Krista Slemmons, Kele Anyanwu, Josh Hames, Dave Grabski, Jeffery Mlsna, Eric Simkins, and Perry Cook. The impact of video length on learning in a middle-level flipped science setting: implications for diversity inclusion. *Journal of Science Education and Technology*, 27(5):469–479, 2018.

[6] Pia Niemelä and Mikko Nurminen. Rate your mate for food for thought: Elsewhere use a grader. In *Proceedings of the 12th International Conference on Computer Supported Education - Volume 2: CSEDU,*, pages 422–429. INSTICC, SciTePress, 2020. ISBN 978-989-758-417-6. doi: 10.5220/0009564204220429.

[7] Sagar Parihar, Ziyaan Dadachanji, Praveen Kumar Singh, Rajdeep Das, Amey Karkare, and Arnab Bhattacharya. Automatic grading and feedback using program repair for introductory programming courses. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, page 92–97, New York, NY, USA, 2017. Association for Computing Machinery.

[8] Abe Leite and Saúl A. Blanco. Effects of human vs. automatic feedback on students' understanding of ai concepts and programming style. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, page 44–50, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367936.

[9] Barry W Boehm, John R Brown, and Mlity Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592–605, 1976.

[10] John Hattie and Helen Timperley. The power of feedback. *Review of Educational Research*, 77(1): 81–112, 2007.

[11] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. Designing a rubric for feedback on code quality in programming courses. 2016.

[12] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. Measuring the effectiveness of error messages designed for novice programmers. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, New York, NY, USA, 2011. Association for Computing Machinery.

[13] Oili-Helena Ylijoki. Akateemiset heimokulttuurit ja yliopistoyhteisön itseymmärrys. *Tiedepolitiikka: Edistyksellinen tiedeliitto ry: n julkaisu 23 (1998): 3*, 1998.

[14] Ville Karavirta and Clifford A Shaffer. Creating engaging online learning material with the jsav javascript algorithm visualization library. *IEEE Transactions on Learning Technologies*, 9(2):171–183, 2015.

[15] Artturi Tilanterä et al. Towards automatic advice in visual algorithm simulation. 2020.

[16] Teemu Sirkiä. Jsvee & kelmu: Creating and tailoring program animations for computing education. *Journal of Software: Evolution and Process*, 30(2):e1924, 2018.

[17] Juha Sorva, Jan Lönnberg, and Lauri Malmi. Students' ways of experiencing visual program simulation. *Comput. Sci. Educ.*, 23(3):207–238, 2013. doi: 10.1080/08993408.2013.807962. URL `https://doi.org/10.1080/08993408.2013.807962`.

[18] Lassi Haaranen and Teemu Lehtinen. Teaching git on the side: Version control system as a course platform. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 87–92, 2015.

[19] Andrea Lucioli. Cloud Native DevOps with Kubernetes and GitOps-a powerfull approach to Continuous Development using Infrastructure as Code. 2020.

[20] Paul Cobb, Jere Confrey, Andrea diSessa, Richard Lehrer, and Leona Schauble. Design experiments in educational research. *Educational Researcher*, 32(1):9–13, 2003.

[21] Peter Reimann. *Design-based research*, pages 37–50. Methodological choice and design. Springer, 2011.

[22] Barbara J. Ericson, Kantwon Rogers, Miranda Parker, Briana Morrison, and Mark Guzdial. Identifying design principles for cs teacher ebooks through design-based research. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, New York, NY, USA, 2016. ACM.

[23] Erkko Sointu, Laura Hirsto, Mari Murtonen, et al. Transforming higher education teaching and learning environments–introduction to the special issue. *International Journal of Learning, Teaching and Educational Research*, 18(13):1–6, 2019. ISSN 1694-2493.

[24] Laura Hirsto and S Väisänen. Exploring the experiences of flipped-learning in a large lecture course in teacher education. In *ECER–conference, Dublin, Ireland*, 2016.

[25] Pia Niemelä, Aulikki Hyrskykari, Timo Poranen, Heikki Hyyrö, and Juhani Linna. Flipped Learning With Peer Reviews in the Introductory CS Course. In *Assessment, Testing, and Measurement Strategies in Global Higher Education*, pages 35–58. IGI Global, 2020.

[26] Feng Wang and Michael J Hannafin. Design-based research and technology-enhanced learning environments. *Educational Technology Research and Development*, 53(4):5–23, 2005.

[27] Terry Anderson and Julie Shattuck. Design-based research: A decade of progress in education research? *Educational researcher*, 41(1):16–25, 2012.

[28] Rikke Ørngreen. Reflections on design-based research. In *Human Work Interaction Design. Work Analysis and Interaction Design Methods for Pervasive and Smart Workplaces*, pages 20–38. Springer, 2015.

[29] The Design-Based Research Collective. Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher*, pages 5–8, 2003.

[30] Jan Van den Akker, Koeno Gravemeijer, and Susan McKenney. Introducing educational design research. In *Educational design research*, pages 15–19. Routledge, 2006.

[31] Allan Collins. Toward a design science of education. In *New directions in educational technology*, pages 15–22. Springer, 1992.

[32] Donald A Schön. Designing as reflective conversation with the materials of a design situation. *Knowledge-based systems*, 5(1):3–14, 1992.

[33] Pekka Koskinen, Joni Lämsä, Jussi Maunuksela, Raija Hämäläinen, and Jouni Viiri. Primetime learning: collaborative and technology-enhanced studying with genuine teacher presence. *International journal of STEM education*, 5(1):20, 2018.

[34] V. Karavirta, P. Ihantola, and T. Koskinen. Service-oriented approach to improve interoperability of e-learning systems. In *2013 IEEE 13th International Conference on Advanced Learning Technologies*, 2013.

[35] M Toivola. Käänteinen arviointi. *Helsinki: Edita*, 2019.

[36] StackOverflow. Stackoverflow developer survey 2020, 2021. URL `https://insights.stackoverflow.com/survey/2020`. [Online; accessed 20-April-2021].

[37] JetBrain. The state of developer ecosystem 2020, 2021. URL `https://www.jetbrains.com/lp/devecosystem-2020/`. [Online; accessed 20-April-2021].