Continuous Analysis of Assignment Evaluation Results from Automated Testing Platform in Iterative-Style Programming Courses

Matej Madeja*, Miroslav Biňas†, Lukáš Prokein‡
Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovakia
*matej.madeja@tuke.sk, †miroslav.binas@tuke.sk, ‡lukas.prokein@student.tuke.sk

***Abstract*—This paper presents a proposal of usage continuous analysis of assignment evaluation results in a real programming course with 585 students to improve syllabus and teacher adaptation. The use of Elastic Stack has proven to be a convenient solution in terms of both efficiency and cost. At the same time the article answers important questions about the attitude of students in such courses. It was found that the structure and compilation issues persist during the whole course, the failure of the first assignment demotivates the student in solving following tasks and the number of students with the highest grades decreases. An important result is that students have only used 10% of the evaluation possibilities on average and work mostly on weekends and in the evenings. At the same time an increased number of above-average results was observed while using the same assignments as in previous course runs.**

I. Introduction

The development of software engineering teaching methods is closely connected to software development practices in real companies. Boehm [1] describes how the view on software development has changed since the 1950s to the present, claiming that the 21st century is characteristic by rapid software development and the introduction of agile methods. The iterative way of teaching can be observed in many massive open online courses (MOOCs) on the internet and universities. Popular methods of development where also included in multiple courses of Technical University of Košice in order to better preparation of students for real companies (see [2]).

There are many ways to motivate students to work continuously on assignments and to be active during labs. Based on the recommendations of Nadipineni [3] and van Vilet [4] we have made changes in several courses at our department, e.g. adjusting course curriculums according to popular online courses, distributing the difficulty of assignments throughout the whole semester and including the human factor in software development (e.g. working in teams). According to Fridge [5] programming of real projects that require real deployment to the production is also a very important factor of a successful course.

In our previous research [6] we observed the impact of the assignment type on student behavior in introductory programming courses at our university through multi-annual analysis of results from the developed testing platform [7]. Although the results have been beneficial it can be assumed that continuous publication of classmates' general statistics can motivate a particular student and improve mental model creation [8]. However, in courses with a large number of students and frequent evaluation of assignments there is a large amount of data that need to be evaluated and published in a short time. This article therefore deals with the proposal for processing and publication of such data at short intervals after evaluation by an automated system.

The data can also help the teacher to respond better to the work of the students. For example, if it is clear that students are not continuously working on the assignment the teacher may change the difficulty of other tasks. Because each course run is attended by unique students who have different experiences, interests and motivations, continuous analysis of students' assignments can improve the learning process for a particular group by tuning the curriculum or adapting the teacher's behavior.

Since at our university the automated testing platform called *Arena* has been collecting data for several years in various courses we would like to to bring new knowledge about the behavior and attitude of the student when developing the assignments during the course. The article answers the following hypotheses:

- **H1:** Problems with the structure and compilation of assignments are highest in the beginning and are gradually decreasing.
- **H2:** Students who gained 0 points for the first assignment improved in the next assignments.
- **H3:** Most students get the best score in problem set 1 (PS1) and this number is gradually decreasing in the next PSs.
- **H4:** The highest activity of students is during weekends and evenings.

978-1-7281-4967-7/19/$31.00 ©2019 IEEE

H5: The average number of used evaluation terms is less than 25%.

H6: Most students achieve an average score.

In Section II *Arena* platform and data visualization options are described. Section III focuses on the design of LST Stack incorporation into existing assignment testing platform, Section IV presents results of the experiment and Section V discuss related work. Research plans for the future and conclusions are described at the end.

II. State of the art

A. Arena platform

Arena[1] is a system for testing and evaluating student programming assignments. The input data is the assignment source code from the university *GitLab* repository. For each assignment the student creates a separate repository with a defined structure and saves his/her solution there. Solutions are evaluated in parallel using the celery[2] workers, each in a separate Docker[3] container. The evaluation runs at specific intervals (e.g. every 3 hrs) and the last commit in the master branch is used. The evaluation process includes the following steps:

- project download from *GitLab* repo,
- unpacking and structure check,
- compilation and testing in a docker container,
- writing results to SQL database and showing them in a web presenter.

Tests execution is sequential. If a test is marked as strict and its evaluation is unsuccessful further tests at that nesting level do not continue. Standard input, output and error output are also part of the test results. The system also offers the possibility to define error messages to help students identify found issues.

Information about the test results of a specific project is in the JSON format and contains information about the test run, project, student, evaluation result and array of each step of the tests. Each testing step contains information about the test, its description, result, type, inputs and outputs or sub-steps.

B. Data analysis and visualization tools

In order to choose the right tools for our solution we made our own ranking of the appropriate tools. The created ranking was based on popularity, research recommendations, price, performance, support, etc. The three best ranked tools from our ranking are briefly compared in the following sections.

Microsoft Power BI[4] is an advanced business data visualization software. In it's free version offers the possibility to import data from available open datasets, unlimited creation of visualizations and their storage in the *Power BI* cloud. Importing custom data by connecting to a live relational or document database is only available in paid packages. *Power BI* does not support Unix-based operating systems (OS).

Tableau[5] offers data analysis from a wide range of sources, from databases to cloud storage. It is available for Windows, MacOS, Linux, iOS and Android. *Tableau* does not have a free version so it requires increased demands on the entry budget. The software is focused on enterprise deployments with an emphasis on collaboration and availability.

Elastic Stack (ELK Stack)[6] is a set of open-source applications for processing, storing, analyzing, full-text search and data visualization. It consists of three main platform components:

- Elasticsearch - realtime distributed search and analytics engine,
- Logstash - tool for collecting, filtering and transforming input data,
- Kibana - tool for data analysis and visualization in the Elasticsearch.

To use *Kibana* as a data visualization tool it is necessary to index the data in the *Elasticsearch* database. All the necessary parts from indexing to data visualization are available in the free version of the *ELK Stack*. The big advantage is the availability of a large community of users and their support in the platform discussion forums. *ELK Stack* can be deployed on own server which is a huge advantage for the usage at university. The platform also has a paid version that adds additional functionality, e.g. authentication, authorization, monitoring, etc.

The most important factor in choosing the final tool were the free version possibilities and OS support. *Elastic Stack* of the above tools offers the best options so it will be used for the solution proposal. *Power BI* is only supported for Windows OS and since Linux servers are mainly available at our university the tool is not suitable. *Tableau* offers only a paid version for which the university does not have the funds.

III. Incorporation of LST stack

Figure 1 shows a conceptual model using the *LST Stack* to visualize students' results from the *Arena* platform. As mentioned in the Section II-A, the input data for the *Arena* is the source code from *GitLab*. This data will enter the *Gladiator-Elastic-Bridge* component which is designed to transform the input data and index it into the *Elasticsearch* database. The data can then be analyzed and visualized using the *Kibana* tool which connects to the *Elasticsearch* database.

Kibana provides the ability to create embeddable visualizations but in free version these visualizations are only functional when the entire *Kibana* is freely available (i.e. without authorization) which is a possible security risk. Therefore, part of the proposed solution is also *Kibana Reporter* module which creates PNG exports of active visualizations. Exports can later be used on foreign websites without mentioned vulnerability issues.

[1] https://arena.kpi.fei.tuke.sk/
[2] http://www.celeryproject.org/
[3] https://www.docker.com/
[4] https://powerbi.microsoft.com
[5] https://www.tableau.com/
[6] https://www.elastic.co/what-is/elk-stack

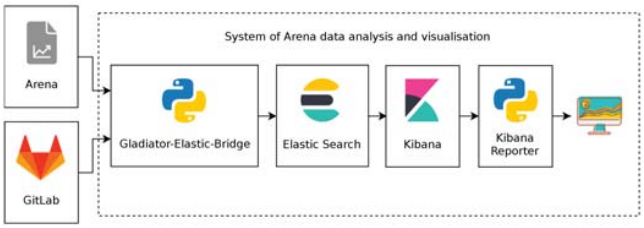

Figure 1. Proposed conceptual model of continuous analysis of students' results.

A. Data preprocessing

Data stored in *Arena* database have JSON format with nested structures. An example of a nested object to be analyzed is the individual `tasks` that belong to a specific `suite`. *Elasticsearch* also supports filtering and aggregation over nested objects. However, when testing *Kibana* possibilities it was found out that it does not support nested objects (see [9]), so visualizations over `task` objects would not be possible with direct indexing data from the *Arena* system to *Elasticsearch*.

This issue has been solved by data transforms using multiple indexes with different mappings without nesting; called flattening. Based on data format in *Arena* database two mapping types were created:

- 1st mapping contains an overall overview of the assignment evaluation, e.g. overall result, info about student and assignment, etc.
- 2nd mapping contains evaluation details of each testing step. Each record contains general information about the student and then details such as gained points, errors, test type, duration, etc.

This whole process is handled by the web microservice *Arena-Elastic-Bridge* which receives POST requests and transforms the data into the necessary form. The prepared data are then inserted into the *Elasticsearch* index using the REST interface. Since the indexing of individual tasks expects a large amount of data the bulk API interface was used which accepts an array of objects in *ndjson* format as the input.

B. Publishing visualizations

Because we want to publish statistics to students continuously during the course and also inform the teacher about the current status of the results, we need to publish the results using *Kibana Reporter*. It is a Python server application based on the use of the *Selenium* API library to work with the browser. Using this library the application opens *Kibana* visualization in Chrome browser in headless mode, waits until the visualization is completely loaded and takes a screenshot in PNG format. This way it takes screenshots of all specified visualizations and using the *scp* protocol copies the resulting images to a web server, e.g. to a specific course website. *Kibana Reporter* runs immediately after evaluating of all students' assignments.

IV. Results

Data from *Programming*[7] in 2018 were used for the analysis. The course was attended by 585 students and we are considering the results of all 4 problem sets (PS); the first three problem sets were mandatory (PS1-3), the last one optional (PS4). In the Table I general statistics about particular assignments in the course can be seen. The assignment evaluation was running at three-hour intervals and the number of days the assignment could be submitted varied according to the difficulty of the assignment.

Table I
General stats about assignments in the course *Programming 2018*.

PS	Involved students	Students with 0p	Commits		Possible evaluations
			max	avg	
1	536	70	368	73	328
2	512	83	114	24	422
3	444	48	151	21	441
4	253	69	109	15	324

PS - Problem Set.

A. Hypotheses verification

The **H1** assumes that when analyzing individual tests and observing unsuccessful test steps the number of failed tests for checking the structure of the assignment, static analysis, and program translation will be the highest at the beginning of the assignment and decreasing to the approaching deadline. In the Figure 2 it is possible to see the number of failures of structure check, static code analysis and compilation results during each day. It can be seen that the structure check failure actually occurred predominantly at the beginning of the assignment, but problems with static code analysis and translation persisted throughout the assignment. The **H1** was therefore refuted.

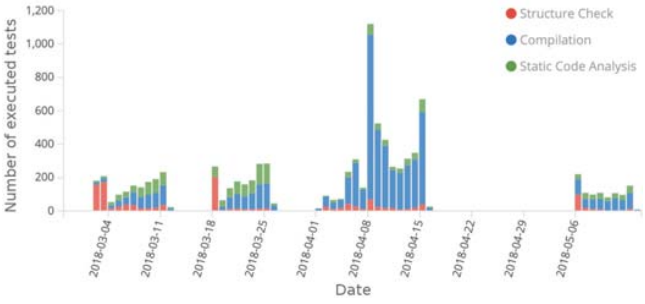

Figure 2. Numbers of failed structure checks, static code analysis and compilations.

This result is interesting because the assignment compilation method was published to the students in advance. Nevertheless, the biggest issues were with the compilation. During the interviews, the students often argued that the compilation mostly failed in cases when the latest solution was submited without thorough tuning, using the "trial and error" approach.

[7]https://kurzy.kpi.fei.tuke.sk/pvjc/

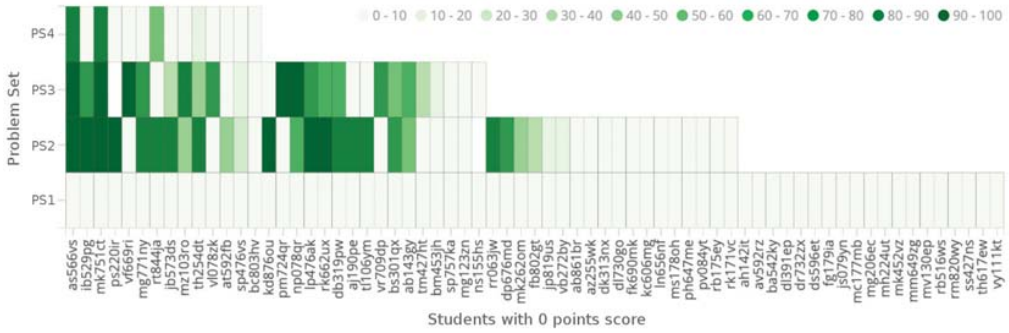

Figure 3. Progress of students' evaluation assessed by 0 points in PS1.

In order to eliminate abuse of this approach the number of student submissions should also be considered in the evaluation in the future. The second students' argument was the opinion that the compilation in a testing (evaluation) environment is different from that in their development environment. At the same time, only the `master` branch was used for evaluation which means that students often develop directly in the main branch and cannot create development branches in the version control system (VCS).

The hypothesis **H2** assumes that if a student gains 0 points in the assignment, the student will work harder in the next assignments and improve his/her results. We analyzed all students who gained 0 points in PS1 and followed their results in other problem sets using heatmap (see Figure 3). It can be seen that the ranking of some students in the following assignments was better, even sometimes achieving excellent results (dark colors of the heatmap). However, improvement occurred in less than 50% of the observed students so the **H2** was refuted.

At the same time we can observe that more than 25% of students did not develop PS2 and this trend continued in further assignments. All PSs were developed by only 20% of these students indicating that the student's first failure has a big influence on further student's work. This fact, however, is unlikely to be affected as it is dependent on the individual characteristics of the student.

To verify the **H3** we monitored the number of students with a 100% evaluation in the PS1. In the Figure 4 it is apparent that the largest number of students with the highest grades is in PS1 and this number decreases during the semester. The decline in success is a good indicator that the difficulty and complexity of tasks increase during the semester. At the same time when comparing the results to the Table I it can be seen that the average number of commits is decreasing. This means that students either work more responsibly and do not use the "trial and error" approach or, on the other hand, are demotivated by the complexity of the assignment. The **H3** was confirmed.

The hypothesis **H4** assumes that students work most on weekends. The Figure 5 shows that students work in addition if they have personal time off, ie. most often over the weekends and Fridays. The highest activity can be observed on Sunday

Figure 4. The number of students with the maximum possible result for each course problem set.

but this result may be influenced by the tendency to develop at the last minute because the deadline for all assignments was on Sunday at 12 PM. Last minute submissions have also been confirmed in our previous analysis [6].

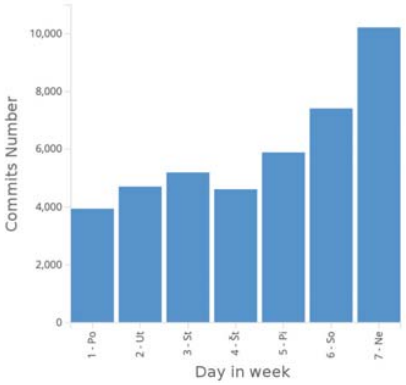

Figure 5. Commits number for each day of the week.

The Figure 6 shows the average number of commits per hour of the day. Most of the student activity is in the afternoon and the highest values are around 8 PM and 9 PM. The **H4**'s claim that students' highest activity is on weekends and in the evening has been confirmed.

Since students' solutions are evaluated every 3 hours and the opportunity to submit the assignment has been at least 7 days in each assignment, students have relatively many opportunities to submit a new solution for evaluation. However, many of the possible evaluations are unlikely not used, as the **H5** claims. It can be seen from the Table II that the average number of evaluation used is far below 25% of the available

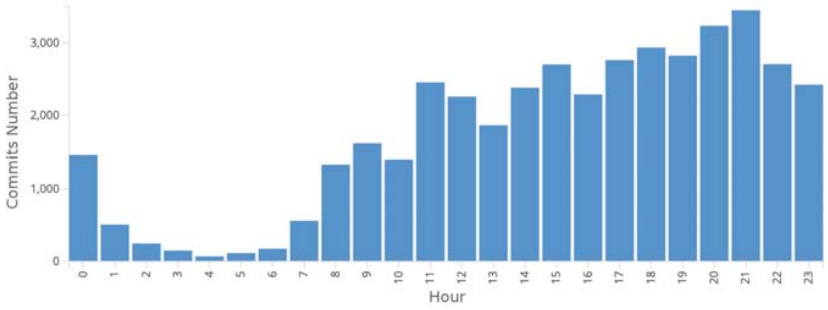

Figure 6. Commits number per hour of the day.

capacity, so the **H5** has been confirmed. At the same time, it was found that the majority of students use the minimum number of commits so as mentioned above it would be useful to develop a dynamic way of assessing assignments, so that students with a large number of submission attempts will be penalized.

Table II
THE AVERAGE NUMBER OF TERMS USED BY STUDENTS IN EACH PS.

PS	Evaluation terms		
	Total No.	AVG/student	AVG$^{-5\%}$/student*
1	93	8.58	8.22
2	76	10.52	10.14
3	116	12.61	12.23
4	60	9.96	9.59

* - Minus 5% of extreme values.

To confirm or refute **H6** we have graphed the overall score for each problem set (see Figure 7). Obviously, with the exception of PS3 where there were the most average students, in other cases there were most above average (PS1, PS2) or below average (PS4) ones. The only difference between the problem sets was that the PS3 contained completely new tasks that had not been used in the course in previous years. The first two problem sets were used in a similar form for the fourth consecutive year, PS4 was used a second time. These results probably indicate that the correct solutions were available to students so PS3 is the most relevant result. It is interesting that during each evaluation we had active anti-plagiarism protection which did not draw so much attention to cheating. This may be due to the fact that the students had the correct solutions, but they were only inspired by them and there was no plagiarism. The hypothesis **H6** was not confirmed.

B. Experience from the teacher's perspective

During the whole semester we tried to actively talk with the students what they liked about the assignments, what tasks they had in other courses and what motivate them to develop or not develop the assignments. When looking at the Table I it can be seen that PS1 and PS2 were attended by the most students. Since PS1 is usually the easiest one students try to work hard from the beginning because they expect more difficult PSs

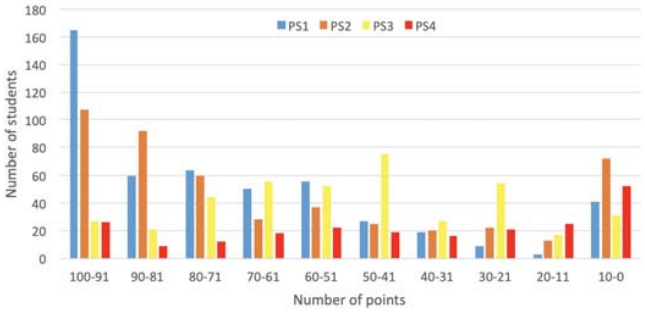

Figure 7. Distribution of students' assignments results in the course *Programming* 2019.

later. In other PSs it can be seen that with increasing difficulty the students are often demotivated and sometimes do not solve the assignment at all because the student reassesses his/her own skills to gain enough points. This decline is often due to a comprehension underestimation of basic programming approaches learned at the beginning of the semester which the students are subsequently unable to use within a more difficult assignment and they prefer to give up. Reduced activity during the assignment is also related to the increased workload of students in other courses at the end of the semester.

As the teachers had all the continuous statistics available they also tried to talk about possible problems with the students. In general, the students did not show frustration with the assignment and they were more enthusiastic, but also aware of the difficulty of the tasks. Compared to previous years course runs we observed a higher number of attempts to evaluate the assignment, ie. students are motivated to work continuously.

The publication of continuous statistics can have a positive impact on students but possible negative impacts should also be considered. For example, if a very small number of students were involved in the first PS and students can see a decline in the interest of their classmates it could also negatively affect their motivation. A student is often inspired by competition among classmates, especially if only a certain number of students can continue their studies. Therefore, from our experience we recommend results that may adversely affect

the student not to publish. Of course, this should be verified by an experiment in the future.

V. Related work

Improving the learning process is subject to continuous development. Blikstein in his work [10] proposes the use of open-ended assignments in programming courses that support improvement of creativity, critical thinking, problem-solving skills, communication and collaboration. Such assignments can evolve in different directions and their evaluation is a complex process. Most often automated systems focus only on assessing outputs and do not consider the whole process of problem solving, e.g. mental model development in the student's head. Blikstein analyzed data obtained during students' assignment solving progress where he recorded what the student writes, where he or her clicks, and especially he observed changes in the code. At the same time, the author suggests metrics that allow to identify possible critical zones during which the student needs help. However, this process is only possible with constant monitoring of the student (e.g. in the classroom), in our case we observed the student's activity at home.

Visualization of data in education also describe Diana et al. in [11]. Their intention is to inform the teacher about possible problems in class work. Students often do not ask the teacher even if they need help. The authors suggest a way to predict such situations and identify students in need of help based on *Alice*[8] system data. The teacher can better respond to the student's needs. In our case we focus on the more global nature of student adaptation, since we focus not only on short duration of labs but on the student's behavior during the whole semester.

Bagui and Fridge [12] describe the benefits of automated systems and their ability to increase student motivation. At the same time, Ihantola et al. [13] analyze various assignment evaluation systems and wonder why others are still evolving. Their claim is that systems usually share the same principles but are mostly designed for one course only. In this paper we use a general system to evaluate assignments in multiple courses and propose the use of this data to better manage courses and motivate students.

Freely available datasets are often used in research, e.g. from the code.org [14] programming competition or the *Code Hunt Game* platform. Perhaps the best known dataset is *Blackbox* [15] which contains compilation level data of hundreds thousands of students but does not contain data on the correctness of the solution. The above researches show that data collection and analysis usually focuses on already ended courses. In our case we try to perform the analysis immediately after the evaluation of all student assignments, ie. in a much shorter time.

VI. Future work

As mentioned in IV-B it is necessary to investigate the impact of publishing negative results on students. At the moment we do not know how students would react to bad results in a real course but our experience suggests that the student is often influenced by the classmates' behavior within the course. At the same time consideration should be given in the future to the possibility of allocating extra points for program efficiency (e.g. CPU usage time) or penalizing too frequent submissions.

In our further research we also would like to focus on comparing the results within different courses, ie. whether the results from the introductory programming courses affect the results in the following ones. It would also be useful to analyze in detail how students make changes in the code during the assignment development. For example, whether the number of new/changed lines in the project is related to the number of points earned, what are the differences in the code if the student tries to submit the assignment in shorter time intervals compared to longer intervals and so on.

Since in this paper we did not focued our observations on interviews with the students in detail it would be advisable to consult current statistics directly with students and monitor their responses to these results. It should be in the interest of the teacher to understand how the students perceive the curriculum, their approach to the course and how the environment influence them (assignments, teachers, classmates or other courses).

VII. Conclusion

In the paper authors describe the integration of continuous data evaluation and visualization possibilities from the automated testing platform *Arena* in programming courses. The *Elastic Stack* was used for the evaluation and the paper also describes a generally applicable design for the evaluation system with used tools description. There are also described methods of data preprocessing for indexing with respect to some limitations of used tools.

Using the newly designed platform in the course with 585 students a number of hypotheses about the behavior of the students in the course are evaluated, as well as the teacher's viewpoints. It was found that the problems with the structure and compilation of the assignment do not occur only in the first problem set but during the whole semester. This is a result of making last minute changes without enough tuning before submitting the assignment for evaluation. It was also found that only 20% of students who gained zero points in PS1 tried to pass on all the other assignments, suggesting that the students are considerably demotivated in the first assignment. On the other hand, the number of students with the maximum number of points decreases during the semester.

This paper also observes when the students most often work on the assignments. It has been found that students work most often in the evening and especially during weekends and Fridays. However, this may be affected by the fact that the deadline was usually on Sunday at 12 PM. On average, the students used only 10% of the possible evaluation terms and only rarely used the "trial and error" approach. It was also found that the students achieved mostly above-average results.

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

The average results were observed only on the assignment that was used for the first time, so above-average results are probably the result of the correct solution inspiration of already existing implementations.

From the teacher's point of view continuous analysis of students' results is beneficial for better course management and dynamic syllabus adaptation. Nevertheless, it is necessary to be vigilant when publishing student results as they may have the potential to decrease student motivation.

ACKNOWLEDGMENT

This work was supported by project KEGA No. 053TUKE-4/2019: Learning Software Engineering via Continues Challenges and Competitions.

REFERENCES

[1] B. Boehm, "A view of 20th and 21st century software engineering," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 12–29. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134288

[2] M. Madeja and M. Biňas, "Implementation of scrum methodology in programming courses," in *2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Nov 2018, pp. 333–340. [Online]. Available: https://doi.org/10.1109/iceta.2018.8572161

[3] N. Nadipineni, "Automated detection of source code plagiarism," Ph.D. dissertation, 2015, copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2016-04-01. [Online]. Available: https://search.proquest.com/docview/1775002265?accountid=49346

[4] H. van Vilet, "Reflections on software engineering education," *IEEE Software*, vol. 23, no. 3, pp. 55–61, May 2006. [Online]. Available: https://doi.org/10.1109/MS.2006.80

[5] E. Fridge, U. of West Florida. Department of Research, and A. Studies, *An Investigation of the Impact of Automated Software Testing Tools on Reflective Thinking and Student Performance in Introductory Computer Science Programming Assignments*. University of West Florida, 2014. [Online]. Available: https://books.google.sk/books?id=-8AHswEACAAJ

[6] M. Madeja and J. Porubän, "Innovative approaches in c introductory programming courses," vol. 2403, 2019. [Online]. Available: http://ceur-ws.org/Vol-2403/paper10.pdf

[7] M. Biňas, "Identifying web services for automatic assessments of programming assignments," in *2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Dec 2014, pp. 45–50. [Online]. Available: https://doi.org/10.1109/ICETA.2014.7107547

[8] M. Sulír and M. Nosál', "Sharing developers' mental models through source code annotations," in *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sep. 2015, pp. 997–1006.

[9] Kibana, "Getting around kibana missing support for nested objects & parent/child," 2017. [Online]. Available: https://discuss.elastic.co/t/getting-around-kibanas-missing-support-for-nested-objects-parent-child/53529

[10] P. Blikstein, "Using learning analytics to assess students' behavior in open-ended programming tasks," in *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, ser. LAK '11. New York, NY, USA: ACM, 2011, pp. 110–116. [Online]. Available: http://doi.acm.org/10.1145/2090116.2090132

[11] N. Diana, M. Eagle, J. Stamper, S. Grover, M. Bienkowski, and S. Basu, "An instructor dashboard for real-time analytics in interactive programming assignments," in *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*, ser. LAK '17. New York, NY, USA: ACM, 2017, pp. 272–279. [Online]. Available: http://doi.acm.org/10.1145/3027385.3027441

[12] S. Bagui and E. Fridge, "Impact of automated software testing tools on reflective thinking and student performance in introductory computer science programming classes," *Int. J. Inf. Commun. Technol. Educ.*, vol. 12, no. 1, pp. 22–37, Jan. 2016. [Online]. Available: http://dx.doi.org/10.4018/IJICTE.2016010103

[13] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '10. New York, NY, USA: ACM, 2010, pp. 86–93. [Online]. Available: http://doi.acm.org/10.1145/1930464.1930480

[14] C. Piech, M. Sahami, J. Huang, and L. Guibas, "Autonomously generating hints by inferring problem solving policies," in *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, ser. L@S '15. New York, NY, USA: ACM, 2015, pp. 195–204. [Online]. Available: http://doi.acm.org/10.1145/2724660.2724668

[15] N. C. C. Brown, M. Kölling, D. McCall, and I. Utting, "Blackbox: A large scale repository of novice programmers' activity," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. New York, NY, USA: ACM, 2014, pp. 223–228. [Online]. Available: http://doi.acm.org/10.1145/2538862.2538924