OBSERVATION OF STUDENTS BEHAVIOUR IN PROGRAMMING COURSES WITH AN AUTOMATED TESTING PLATFORM AT DIFFERENTLY GEOLOCATED UNIVERSITIES: A CASE STUDY

Matej MADEJA*, Jaroslav PORUBÄN**, Veljko PEJOVIC*** Martin GJORESKI****

*,**Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovakia
E-mail: matej.madeja@tuke.sk, jaroslav.poruban@tuke.sk

***Faculty of Computer and Information Science,
University of Ljubljana, Vecna pot 113, 1000 Ljubljana, Slovenia,
E-mail: veljko.pejovic@fri.uni-lj.si

****Jozef Stefan Institute, Department of Intelligent Systems
Jozef Stefan Postgraduate School, Jamova cesta 39, 1000 Ljubljana, Slovenia,
E-mail: martin.gjoreski@ijs.si

ABSTRACT

One of the main tasks of a course is that it should motivate students to work continuously during the semester. Using the same course curriculum for students with different cultural backgrounds can have different impacts on student behavior and motivation. In this paper is conducted a case study at Technical University of Košice and University of Ljubljana, where both universities used the same course curriculum, automated testing environment (ATE) and assignment to observe student behavior in a programming course. It was found that by shortening the period of testing days in ATE students will not significantly affect the amount of its continuous work. On the other hand, shortening this period has negatively impacted student score which can be frustrating for the student. Another finding was that on average 58% of submissions ends with failed build. Based on the manual testing it can be stated that in the course with the ATE students understand the topics of Android development better and devote the course 1.7 times more time than in the course without the ATE.

Keywords: *Automated testing environment, student motivation, Android programming course, eduscrum, commit frequency, university comparison.*

1. INTRODUCTION

The challenge of learning programming is as old as programming itself. Many teaching researchers are looking for appropriate approaches for more effective student involvement [1] and ways to motivate students. Jenkins [2] already in 2001 talks about student motivation arguing that motivation is an instructor's key role. Although the personality of the instructor may motivate students, especially at lectures and labs, the instructor does not have sufficient influence on the students outside the university.

The business and technology development also influence course and students' enthusiasm to the course. Nowadays, when technology development grows very fast it is difficult to meet students' ideas about a particular language or platform. If the student's motivation in the course is based only on the technology that is currently popular, it may happen that the technology will not exist after his/her graduation. Although technology itself has a high importance to the student, especially when choosing a course, but it is preferable to motivate him or her by modern software development methods (SCRUM, Extreme Programming, etc.), similarly as to motivate employees in practice.

Many teachers share the opinion that games have a positive impact on student's activity but this approach is better suited for introductory programming courses [3]. In general, teaching programming using only games is too domain-specific because students do not learn solving real problems in classic projects. Mohorovičić and Strčić [4] describe in more detail how other strategies can positively influence the teaching of programming. Authors of this paper think that the syllabus of the course has the greatest impact on student motivation and students can also motivate each other, e.g. using challenges and competitions.

As we have found out in our previous research on motivation continuous delivery [3] with the support of automatic testing environment [5] really positively influence the motivation (not only in education). Most of the experiments carried out in this research area were conducted only within one university or company and often with a limited number of participants. This paper focuses on using a similar course syllabus with the same semestral project and testing platform at two different universities placed in different regions. The following hypotheses are answered:

H1: The average difference of the automated test results of a particular sprint varies by less than 10% between universities.

H2: The average commit rate per user is much higher if students have fewer terms to submit their solutions.

H3: More than 30% of student submissions end with `BUILD FAILD` error.

H4: The students' opinions on automated testing are negative.

H5: The number of hours worked in the course with an automated testing environment (ATE) is higher than in the course without an ATE.

In the following section the universities, courses, participants, semestral project and used methodology are described. In the Section 3 used methods for data acquisition and their analysis are discussed and in the Section 4 results are presented. Threats to validity are pointed out in the Section 5, related work in the Section 6, and conclusions and future work are presented in the Section 7.

2. BACKGROUND

2.1. Universities, courses and participants

In the case study two universities are included: *Technical university of Košice* (TUKE) and *University of Ljubljana* (UNI-LJ). They are about 750 km away and thus represent a different sample of people involved in the experiment.

The course *Application development for smart devices* in fall 2018 was used at TUKE. The course focuses exclusively on the development of applications for smart devices on *Android*, *iOS* and *Windows phone* platforms. ATE was used only for 161 native *Android* projects. Participants were 3rd year undergraduate students and the course had object-oriented programming (OOP) and Java courses prerequisites. The results from ATE represented a major part of the final evaluation of students, supplemented by the results of manual testing. Hackathon participation was also a condition for passing the course.

The spring 2019 run of the course *Platform Based Programming* was used at UNI-LJ which initially focused on Adruino platform development, but in the global viewpoint focused primarily on native *Android* application development. The course was attended by 93 undergraduate students. The evaluation consisted of the following:

- 50% Coursework, out of which:
  - 10% Homework assignments (multiple smaller tasks)
  - **90% Class project** (main assignment), out of which:
    - 15% Project proposal
    - 35% Mid-semester presentation
    - 50% Final presentation and report
- 50% Final exam

The class project/assignment is the part of the course that is analyzed. For a partial comparison we used fall 2019 *Application development for smart devices* course but with a bigger change of the curriculum. The focus in 2019 was to develop hybrid applications in the *Cordova* javascript framework and without use of automated tests.

Detailed information about the courses can be found on UNI-LJ course web page[1] or TUKE course web page[2].

2.2. Real project as assignment

The main assignment of courses in fall 2018 and spring 2019 was the sports monitoring and tracking application called *Makacs*. The application was chosen because of the necessary functionality of this type of applications, including:

- Sensors - GPS, accelerometer, gyroscope.
- Services - stopwatch as background task, widgets.
- Third party services - firebase, back4app.
- Intents - optimalisation of data sending between activities.
- Design and UX - own design proposal and testing.

Only the most interesting areas within the app was listed. As can be seen, it was possible to cover most topics of Android development within one application, so the application is suitable for use in such course. Requested minimal API level was 23+ and there were no other implementation restrictions (except for needed testing environment ones). A more app description can be found in [5] and full specification is published on the official TUKE course web page[2], archive section (slovak language).

2.3. Course development methodology

The agile SCRUM method was used in the course to lead students' projects and the development team consisted of only one student. In this way every student has to work on all topics of the assignment. In the real team often happens that only a few students actually work on the assignment and the others are parasites.

We divided the semester project into 4 sprints:

- Sprint 1 – application design.
- Sprint 2 – minimum viable product (MVP) version.
- Sprint 3 – big change in the project specification, final requirements.
- Sprint 4 – enriching application by own functionality.

ATE was used for all sprints except the 4th sprint which was tested only manually as students were supposed to implement their own original functionality. In the 3rd sprint we made a major change in the functionality of the project to simulate customer's unexpected product changes. In this way we try to lead students to increase code sustainability where they also tried refactoring on a real product (assignment). At both universities, a 2-hour dedicated consultation lab was held at the beginning of each sprint where the sprint specification was presented to the students and also the trickier requirements were discussed more precisely. During the sprint students could request individual consultations but most of the issues were consulted online via *Slack*[3]. More information about this modified version of SCRUM can be found in [3].

[1] http://lrss.fri.uni-lj.si/Veljko/pbd2019.html

[2] https://kurzy.kpi.fei.tuke.sk/smart/

[3] https://slack.com/

3. METHODS

To draw the most accurate conclusions about specific course runs we drew information from various sources. The motivation of the student depends not only on the type of assignment and the syllabus, therefore it is necessary to consider his/her opinions about the course. Observations in this paper depend on ATE data, manual testing and questionnaires.

3.1. Different sprint and testing duration

The sprints at each university were of varying length in order to observe the impact of the sprint length and the number of testing days on student activity. In the Table 1 general statistics about the sprints can be seen. In the 1st sprint tests were performed only once because it was only a UI elements check. TUKE also evaluated the quality of the design by manual evaluation in the first sprint. Since only the existence of screenshots was verified at UNI-LJ we will only consider the existence of screenshots for equitable comparison of results. The 4th sprint was fully manually tested so some data are not listed.

Table 1: General statistics about sprints' duration in the particular course.

#	Duration in days				Number	
	Sprint		Testing		of evaluations	
	TUKE	UNI-LJ	TUKE	UNI-LJ	TUKE	UNI-LJ
1	14	14	1	1	1	1
2	14	21	6	21	7	63
3	21	28	15	21	16	49
4	21	19	-	-	-	-

# - Sprint number.

As can be seen in the Table 1 the number of testing days was always less than the length of the sprint. In this way students were motivated not to use the trial-and-error method to achieve a good result but they had to look more closely at the specification. At UNI-LJ the number of days with testing was less stringent. By combining this data it is possible to compare whether students are performing better if the testing environment is available for longer time.

3.2. Automated testing environment

Students submitted their solutions via version control system (VCS) *git*[4]. At TUKE a private department's *GitLab* and at UNI-LJ *Bitbucket*[5] were used. This method of code submitting not only facilitates the management of project changes for the student but also provides valuable information for the teacher - it is possible to analyze exactly when the student commits his/her changes, it is possible to test different versions of the project/branches and we can observe how the student manages his/her changes (detection of trial-and-error method, cheating, etc.). In our case only the `master` branch was used for testing. The project was only tested if there was a change in the source code since the previous test run. From the mentioned *git* platforms we also collected exact commit times with the number of changed files, insertions and deletions. The detailed structure and the whole process of the testing process are described in more detail in [5].

Automated tests were run at predefined intervals. For TUKE there were planned times shown in the Table 2. In fact, some of the tests were performed a few hours later (max. 6 hours delay) due to test errors or other discrepancies found. However, during 2nd sprint most of the problems were solved so in 3rd sprint testing times were accurate. Although test results and test runs may have been delayed by several hours the student code was always tested from the time of scheduled testing.

Table 2: Testing times at TUKE for the particular sprint.

#	**Automated testing times**
1	2018-11-04 00:00
2	everyday at 12:00 from 2018-11-19 to 2018-11-24 additional testing times: - 2018-11-23 00:00:00 - 2018-11-24 06:00:00 (extended deadline by 6 hours)
3	everyday at 06:00 from 2018-12-15 to 2018-12-29 additional testing times: - 2018-12-15 19:45 - 2018-12-16 18:00

# - Sprint number.

At UNI-LJ were tests also run at predefined intervals but in most cases they were performed 3 times a day. The Table 3 shows the timetable of the planned test execution. The results were available to students mostly up to 2 hours, during a higher ATE load (mosty at 23:30) of up to 8 hours.

Table 3: Testing times at UNI-LJ for the particular sprint.

#	**Rule**
1	2019-04-01 0:00
2	everyday at 12:00, 20:00 and 23:30 from 2019-04-01 to 2019-04-21

3	everyday at 23:30 from 2018-04-29 to 2019-05-05;
	everyday at 12:00, 20:00 and 23:30
	from 2019-05-06 to 2019-05-19

# - Sprint number.

3.3. Manual testing

Because there are ways to write a program that satisfies the specification it is necessary to perform a manual check that the application actually meets the requirements. It is therefore not appropriate to rely solely on the ATE in this case. The tests should look for errors in the code from which some functionality is expected and want to prevent errors. When using such an environment in university courses often leads the student only to pass the tests, so in some cases he or she is able to write such a code to get more points.

[4] https://git-scm.com/

[5] https://bitbucket.org/

Manual testing was performed only in the 4th sprint as students programmed their own original functionality. At TUKE and UNI-LJ the overall usability of the application was evaluated and the functionalities from previous sprints were checked. With manual testing the teacher can see how a student responds to questions, what he/she has problems with, how he/she know the source code and, finally, whether the program presented the student is created by himself/herself. The overall score was calculated by a combination of automated and manual testing where manual testing was a kind of ”confirmation” that ATE tested a real functionality.

3.4. Questionnaire

At the end of semester a questionnaire was used to collect opinions about the course which consisted of the following items:

1. Write at least 3 positive aspects of the course.
2. Write at least 3 negative aspects of the course.
3. How many hours did you spend on the course? (incl. lectures, labs, assignment programming, testing and any individual work)
4. Some additional notes to the course.

There were only open answers because we did not want to limit the student’s answers (e.g. to rate course quality using a scale of 1-5 points). Processing results of a questionnaire is often much easier to evaluate if a scale was used, but such questionnaires often fail to capture the student’s overall view.

4. RESULTS

In this case study we analyzed 9845 of commits and 64327 test results. General statistics of included courses are shown in the Table 4. As can be seen, the average ATE result differs by 18.8%. If we only rely on ATE results, TUKE students would fail the course (under 50% of overal score). Only the best students’ result in the particular sprint was always included into the final score. All results can be downloaded from Github[6].

Table 4: General stats about courses.

University	Course run	Students	Avg. ATE score
TUKE	fall 2018	161	45.53 %
	fall 2019*	166	-
UNI-LJ	spring 2019	93	64.33 %

* - Course run without ATE.

4.1. Automated testing environment

The first hypothesis **H1** asserts that the difference between the results of different universities using the same ATE, tests and assignment will be less than 10%. To answer the **H1** a comparison of the average results of each sprint for a particular university was created (see the Table 5). It can be seen from the table that only the first sprint had a difference in average rating of less than 10%, therefore, **H1** was rejected.

Table 5: Avg. sprint score of universities.

	University	Avg. result	Diff
Sprint 1	**TUKE**	89.31 %	9.21 %
	UNI-LJ	98.90 %	
Sprint 2	**TUKE**	61.45 %	18.74 %
	UNI-LJ	70.58 %	
Sprint 3	**TUKE**	33.24 %	29.83 %
	UNI-LJ	49.94 %	

Because a large number of 0 point students were observed in the data the Table 6 was created, showing the number of 0 point students in each sprint and the average result without these students. From the data is clear that this large difference in average ranking between universities was due to the fact that most students submitted the 1st sprint, but a large number of students did not submit 2nd and 3rd sprint. Therefore, we have temporarily removed these students and the difference between the average result of universities has narrowed slightly (see the Table 6). Thus, the average result of TUKE was negatively influenced by students who did not continue the course or just skipped 3rd sprint because they had enough points to pass the course from previous ones.

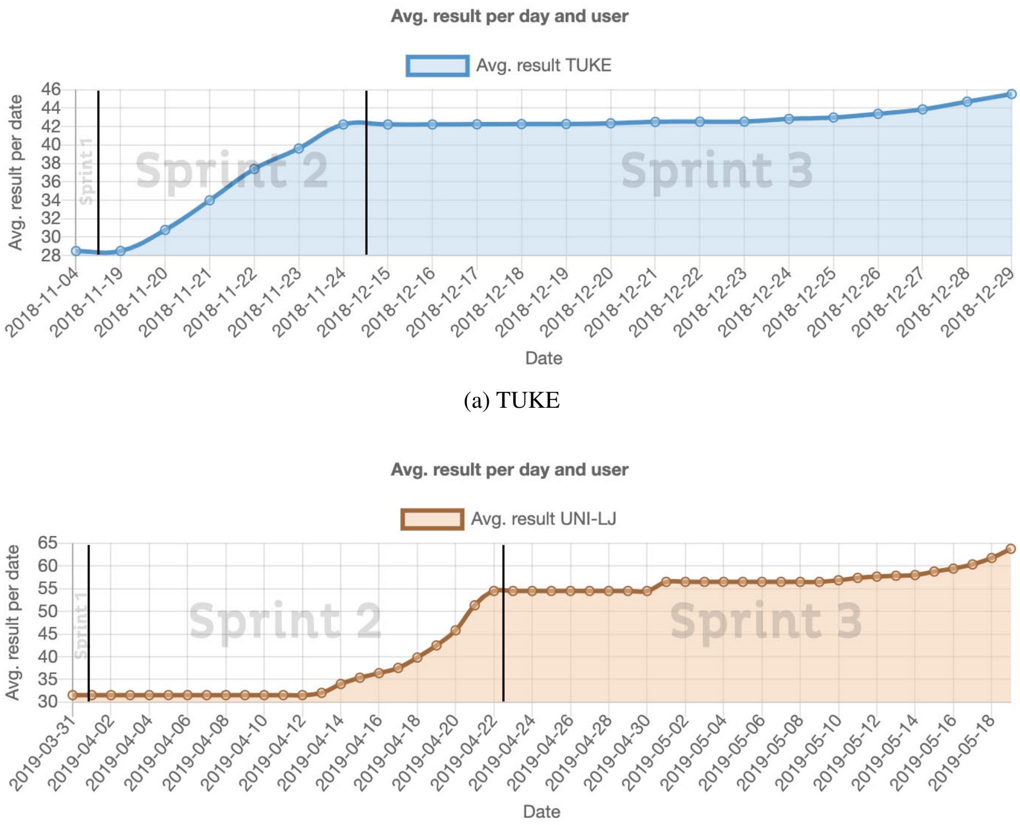
As can be seen, both universities had the largest number of students with 0 points in the 3rd sprint and at TUKE the growth of these students was rapid. The situation probably arises because of the need to develop assignments for other courses. On the other hand, the particular sprints tended to increase in difficulty so it could also deter students from developing. In the Figure 1 a detailed evolution of the average score over the testing days is shown. Deflection of the 3rd sprint at TUKE predominantly affected the results. Despite more testing days at UNI-LJ students did not improve their results (beginning of the 2nd and 3rd sprint, see Figure 1b). If we do not consider mentioned terms then results improvment at both universities has a similar upward trend.

Table 6: Avg. sprint results without 0 point students.

	University	No. of 0p students	Avg. result*
Sprint 1	**TUKE**	7	89.31 %
	UNI-LJ	4	98.9 %
Sprint 2	**TUKE**	53	61.45 %
	UNI-LJ	14	70.58 %
Sprint 3	**TUKE**	113	33.24 %
	UNI-LJ	19	49.94 %
Final result without 0p students in sprint 2 and 3		**TUKE**	53.66 %
		UNI-LJ	69.90 %

* - Without 0 point students.

[6] https://github.com/madeja/programming-courses-observation



(a) TUKE

(b) UNI-LJ

Fig. 1 : Evolution of the average score over each testing day.

The **H2** hypothesis assumes that if a student has fewer submission deadlines more work will be done between testing rounds. This work can be tracked by the number of commits that the student performs between each submission. It is assumed that the fewer options a student has to submit his/her solution the more commits will be done because he/she will try to pass more fixes in one submission.

In the Figure 2 a comparison of the number of commits per particular day and student can be seen. The closer the deadline is the more motivated the student is to submit the best possible solution to reach as many points as possible. The university graphs were aligned just with respect to the deadline to make them easier to compare. It can be seen that TUKE students have always made fewer commits. However, this data is also influenced by the fact that UNI-LJ tested solutions normally 3 times a day while TUKE mostly once a day. Finally, TUKE students have used a greater number of available testing terms.

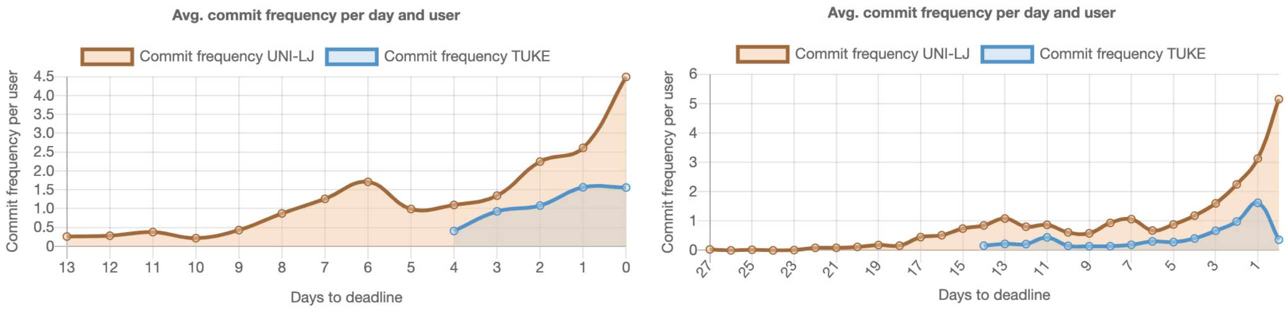
On the other hand, when considering the number of average changes made in each commit (see Table 7), it can be seen, that TUKE students were more cautious about the changes. This was probably due to the fact that if students had fewer opportunities to submit their assignments they were afraid to make major changes because of the greater risk of compilation failure. From these results it can be argued that fewer testing terms motivate the students to make smaller and more thoughtful changes to get rather less points but with less risk of compilation failure. The hypothesis **H2** was rejected.

Table 7: Average changes in commits.

Parameter	**University**	**Mean**
changed files	TUKE	146
	UNI-LJ	227
insersions	TUKE	4982
	UNI-LJ	8663
deletions	TUKE	460
	UNI-LJ	322

During the consultations at both universities a frequent problem with the compilation of the students' solutions were observed. For the testing process the necessary identifiers, test method names, etc. have been defined and students had to fulfil them for successful compilation process. If a student did not fulfil any of the requirements it often resulted in a failed build. Thus, the **H3** hypothesis assumes that the total number of compilation failures is greater than 30%.

The compilation for unit and UI tests was executed separately because only the necessary tests were always packaged in the compilation. Therefore, it was possible that unit



(a) Sprint 2

(b) Sprint 3

Fig. 2 : Average commit frequency per day and user.

tests were compiled but UI tests did not and vice versa (e.g. unit tests did not use the identifier that was needed for UI tests compilation). In the Table 8 can be seen that the number of failures is greater than 40%. The average number of failures for TUKE was 68% and for UNI-LJ 48 % which represents nearly 20% difference between universities. It should be noted that only solutions with changes were sent for testing (not all projects were always tested). The hypothesis **H3** was proven.

Table 8: The amount of build failures for particular university.

#	University	unit	UI	No. of submissions per student
2	**TUKE**	49.71 %	67.13 %	5.25
	UNI-LJ	42.99 %	54.28 %	11.43
3	**TUKE**	76.67 %	80.1 %	3.76
	UNI-LJ	46.14 %	48.96 %	12.96

# - Sprint number.

4.2. Manual testing

Application Development for Smart Devices has been taught at TUKE since 2015 and only in 2019 the ATE was used as the main source of student assessment. Despite the much worse results compared to UNI-LJ, we can say from the observations during manual testing that students understood topics and their relationship in Android development in more detail. Students were able to respond better to questions about their code and understood all the issues much better than without use of the ATE. Although students may had a negative ATE experience (e.g. rigorous tests, high number of builds failures) they worked much more to solve issues than when they created the application in their own way without detailed guidance (compared with previous course runs).

At UNI-LJ the assignment and teaching of Android development was included for the first time, so it is not possible to compare the impact on students to such an extent. Nevertheless, the ATE experience has proven successful and tests have led students to a more stable solution.

4.3. Questionnaire

At the end of each course a questionnaire for the courses was sent to all students. At TUKE we also included the 2019 answers where the ATE testing was not performed. A total of 168 responses were collected.

To answer the **H4** hypothesis we manually read all the students' answers because the questionnaire contained opened questions to get the best feedback. The hypothesis assumes that students' opinion about the course is negative when ATE tests are performed in the course. We categorized each answer into groups and observed only the following 2 attributes:

1. Negative feelings about the ATE.
2. Nagative feelings about assignment workload (too much work opinion).

As can be seen in the Table 9 negative feelings about ATE are higher at TUKE and conversely, negative feelings about assignment workload is higher at UNI-LS. Both of the feelings are above 40% in average for both universities so the overall impression of the course is mostly positive, but the negative aspects are close to half. However, **H4** was rejected.

Table 9: Avg. sprint results without 0 point students.

Property	**University**		
	TUKE		**UNI-LJ**
	fall 2018	**fall 2019***	**spring 2019**
Answers	62	81	25
NFa* ATE	53 %	-	32 %
NFa* workload	26 %	0 %	56 %
Spent hours**	96.42	57.75	103.95

* - NFa = Negative feeling about; ** - during whole course

On the other hand, to confirm or reject the **H5** hypothesis the Table 9 can be also helpful. It can be seen that the number of students working in an ATE course is approximately 1.7 times higher than in a non-ATE course. Students consider the testing platform as one of the biggest negatives but at the same time it motivates them to work on the assignment more continuously and they better understand the basics of Android (compare with the Section 4.2). Their continuous results are worse because the tests are much more strict than the manual evaluation, on the other hand, they are fairer. The **H5** has been confirmed.

During the questionnaire responses evaluation we encountered a common problem of identifying errors from logs. Students should already have debugging experience from previous courses but they probably don't have enough experience yet, so more time should be devoted to this topic in previous courses.

In addition to negative responses there were also positive responses to the application that had to be implemented as an assignment. The greatest benefits were attributed to students by its usability and the amount of topics they met. On the other hand, in the course at TUKE in 2019 we encountered a negative feedback on the freedom to choose the application the student will implement. They consider manually evaluation of various assignments as unfair.

5. THREATS TO VALIDITY

The case study was carried out at 2 European universities. In order to generalize the results, it would be necessary to use the same course in other countries and also outside of Europe. The experiment was also conducted at universities that are free of charge for native full-time students. In the future it would be appropriate to compare the motivation of students of paid programs.

The course was taught in slovak lenguage at TUKE while only a few groups of foreign students were taught in English. The course scenarios and Slack communication were in Slovak but the entire definition of the assignment was in English. At UNI-LJ the whole course was conducted in English. The use of a foreign language in the course may have influenced the results because a language barrier could have arisen. Also the diversity of students (native vs. foreign) could influence the results because the majority of the students were native.

To make the results even more accurate it would be advisable to monitor the overall work of the students in the project. Using the questionnaire we obtained information on the time required for the given respondent but this assessment may be quite inaccurate. It would be appropriate to monitor the respondents' activity, e.g. using a suitable IDE plugin. Using the plugin it should be possible to obtain more detail information about the functionality the student has a problem with and could be adapted to the syllabus.

The testing terms at universities were not the same. E.g. UNI-LJ testing was performed 1-3 times a day, at TUKE mostly once a day with a few additional testing times (max. 2 times per day). To make the results more comparable it would be appropriate to set the same testing frequency for all universities. The comparison of the questionnaires at TUKE between 2018 and 2019 was carried out at the same rate but in 2019 the hybrid development has been used instead of native one. The difference in the development framework used could undesirably influence feedback.

In the analyzes information from specific commits was used but the fact that the code did not have to be immediately pushed to the server was not taken into account. The real push of the code to GitLab could be executed later, so it would be appropriate to consider this attribute in the future.

6. RELATED WORK

Similar experiment as described in this paper was conducted by Johnson [6] in 2015. Author analyzed 185 university students by questionnaire of those who reported studying on-campus with those who reported studying fully-online. Students who studied fully-online were older, more likely to be native English speakers and had lower expectations of academic achievement. On-campus students had higher levels of extrinsic achievement motivation and expressed greater need for peer and teacher support for learning. This paper is more focused on real students' behaviour in the course and his/her activity. We are also not limited to one university but we compare the use of the same method/syllabus at universities with different geographical locations.

Interactive e-learning environments are increasingly used and Navrat and Tvarozek [7] proposed the way how students can be assessed for grades by such environments. Using data from three different university courses they calibrated the two parameter logistic regression model, ranked students according to their ability of solving problems, and matched them to final grades. Results indicate it is possible to predict grades within 0.57 to 1.02 level of accuracy. In our case, we do not try to predict the results or assessment but we analyze submissions and try to find the differences between students of different universities using the same teaching method. The results could support further modification of the proposed method and syllabus.

Devadiga [8] observed gaps in converging software engineering education with the startup industry. He monitored the practices used in startups and how they affect the involved student. According to him, in the education of software engineering, the student focuses mainly on functionality and forgets about proper planning, does not think about architecture design, underestimate code reviews and so on. In the analyzed courses we tried to minimize these gaps by using the testing environment and continuous delivery of submissions. In addition, we analyze the behavior of students and partially compare the time demand with the same course without the ATE.

Vahldick et al. [9] analyzed list of 40 games in university education, classified them by type and highlighted the skills and topics supported by them. As we have mentioned before, games bring a high motivation level for students and, according to authors, students engage in that work and do not get frustrated with the natural errors they will make in this process. Because this paper observed students motivation in more general assignment and it focused more on methodology of the course, it analyze and provides a generally applicable solution in software engineering education courses.

7. CONCLUSIONS AND FUTURE WORK

This paper analyzes the students' behavior and motivation at 2 different universities in Slovakia and Slovenia. The same automated testing environment (ATE) was used in the courses to test student solutions of a sport tracking application. Collected data from ATE were subjected to analysis which included data from VCS *git*, build and testing results from every testing round and student. By changing the length of time the solution could be submitted for testing, the difference in student results and activity was monitored. At the same time, the paper takes into account the results of observations during manual testing and the opinions of students obtained through a questionnaire.

It has been found that shortening the deadline for submitting the assignment to ATE has a negative impact on the student, which may discourage him from working in subsequent sprints. Student score for the 1st sprint varied by approximately 9%, in the 2nd sprint 19% and in the 3rd sprint 30%. The final average score was 19% worse at TUKE and this result was mainly due to the large number of 0 point students in the 3rd sprint, which was the most challenging and complex.

Student activity due to higher number of testing rounds during a day was higher at UNI-LJ. Nevertheless, the growth of average number of commits was similar to the approaching deadline (the graph had a similar rise). However, the duration of testing did not affect the improvement of students' results as the results began to improve as the deadline approached. In terms of results there is no need to allow a longer run of the testing platform. Finally, it was found that at UNI-LJ 48% and at TUKE even 68% of submissions end with build failure.

From the manual testing experience in courses with and without the ATE, it can be stated that students understand better the topics of Android development in the course with the ATE. This is due to the longer time needed for debugging and motivation to pass the tests. Although nearly half of students have a negative experience with the ATE, they spend 1.7 times more time than in a non-ATE course, so they also have more experience with possible issues during the development of native application for Android platform.

As the Android mobile app development course has been shown to be too challenging because Android development is changing incredibly every year, we tried to change course design and apply hybrid development. In the future we would like to use a similar testing method for a hybrid development course using the created platform at various universities around the world or in MOOC courses. Gaining results from different geological parts can lead us to better curricullum management according to the individual needs of the students in the course.

ACKNOWLEDGEMENT

The authors would like to thank all the participants in the research, especially students and teachers of courses *Application Development for Smart Devices* at Technical university of Košice and *Platform Based Programming* at University of Ljubljana. This work was supported by project KEGA No. 053TUKE-4/2019: Learning Software Engineering via Continues Challenges and Competitions.

REFERENCES

[1] A. Vihavainen, J. Airaksinen, and C. Watson, "A systematic review of approaches for teaching introductory programming and their influence on success," in *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14, (New York, NY, USA), pp. 19–26, ACM, 2014.

[2] T. Jenkins, "The motivation of students of programming," in *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '01, (New York, NY, USA), pp. 53–56, ACM, 2001.

[3] M. Madeja and M. Biňas, "Implementation of scrum methodology in programming courses," in *2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pp. 333–340, Nov 2018.

[4] S. Mohorovičić and V. Strčić, "An overview of computer programming teaching methods," in *XXII Central European Conference on Information and Intelligent Systems*, pp. 1–6, 2011.

[5] M. Madeja and J. Porubän, "Automated testing environment and assessment of assignments for android mooc," *Open Computer Science*, vol. 8, no. 1, pp. 80–92, 2018.

[6] G. M. Johnson, "On-campus and fully-online university students: Comparing demographics, digital technology use and learning characteristics," *Journal of University Teaching & Learning Practice*, vol. 12, no. 1, p. 4, 2015.

[7] P. Navrat and J. Tvarozek, "Online programming exercises for summative assessment in university courses," in *Proceedings of the 15th International Conference on Computer Systems and Technologies*, CompSysTech '14, (New York, NY, USA), pp. 341–348, ACM, 2014.

[8] N. M. Devadiga, "Software engineering education: Converging with the startup industry," in *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)*, pp. 192–196, Nov 2017.

[9] A. Vahldick, A. J. Mendes, and M. J. Marcelino, "A review of games designed to improve introductory computer programming competencies," in *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pp. 1–7, Oct 2014.

Received December 23, 2019, accepted February 24, 2020

BIOGRAPHIES

Matej Madeja was born in 1992 in Kežmarok, Slovakia. In 2017 he graduated (MSc) at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University of Košice. He defended his master's thesis in the field of Informatics. Currently, he is a PhD student in the same department. His research is focused on improvement of program comprehension efficiency, source code testing techniques and environments, and teaching of programming.

Jaroslav Porubän is an Associate professor and the Head of Department of Computers and Informatics, Technical University of Košice, Slovakia. He received his MSc. in Computer Science in 2000 and his PhD. in Computer Science in 2004. Since 2003 he is a member of the Department of Computers and Informatics at Technical University of Košice. Currently the main subject of his research is the computer language engineering concentrating on design and implementation of domain specific languages and computer language composition and evolution.

Veljko Pejovic is an assistant professor at the Faculty of Computer and Information Science (FRI), University of Ljubljana, Slovenia. He received his PhD. in Computer Science in 2012 at University of California Santa Barbara, USA. His research deals mainly with mobile computing. Lately he has been focused on mobile sensing, approximate mobile computing, and the security in the IoT domain.

Martin Gjoreski is a PhD student at the Department of Intelligent Systems, Jozef Stefan Institute, Slovenia. His research focuses on monitoring human psychological and physical states using wearables. He received his MSc in Information and communication technologies from the Jozef Stefan Postgraduate School in 2016.