Automated Assessment of Android Exercises with Cloud-native Technologies

Daniel Bruzual
Eficode
daniel.bruzual@eficode.com

Maria L. Montoya Freire
Aalto University
maria.montoyafreire@aalto.fi

Mario Di Francesco
Aalto University
mario.di.francesco@aalto.fi

ABSTRACT

Mobile applications are very challenging to test as they usually have a complex graphical user interface and advanced functionality that involves interacting with remote services. Due to these features, student assessment in courses about mobile application development usually relies on assignments or projects that are manually checked by teaching assistants for grading. This approach clearly does not scale to large classrooms, especially for online courses. This article presents a novel system for automated assessment of Android exercises with cloud-native technologies. Different from the state of the art, the proposed solution leverages a mobile app testing framework that is largely used in the industry instead of custom libraries. Furthermore, the devised system employs software containers and scales with the availability of resources in a data center, which is essential for massive open online courses. The system design and implementation is detailed, together with the results from a deployment within a master-level course with 120 students. The received feedback demonstrates that the proposed solution was effective, as it provided insightful feedback and supported independent learning of mobile application development.

CCS CONCEPTS

• **Social and professional topics** → **Student assessment**; • **Applied computing** → *Interactive learning environments.*

KEYWORDS

Android; mobile app development; software containers; UI testing; full-stack; automated grading; online learning; computer science education

ACM Reference Format:
Daniel Bruzual, Maria L. Montoya Freire, and Mario Di Francesco. 2020. Automated Assessment of Android Exercises with Cloud-native Technologies. In *2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'20), June 15–19, 2020, Trondheim, Norway.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341525.3387430

1 INTRODUCTION

Online learning has seen a large growth in recent years. Its scope ranges from professional training to remote education, primarily at the university level. The advantages of online learning include lower delivery costs, more flexibility in student participation, and easy scaling to a large number of enrollments [17]. Online learning is also increasingly used for courses given at traditional (i.e., non-virtual) universities, as a complement to contact sessions and means to increase student engagement outside the classroom.

Mobile application development is an important subject in computer science education, due to the pervasive availability of devices such as smartphones and their prevalence in accessing a variety of Internet-based services [10]. Furthermore, companies demand more and more of related skills, especially for fast-growing businesses in diverse operating sectors. Therefore, online activities related to mobile application development offer several opportunities to both students and institutions [15].

Unfortunately, developing mobile applications is also challenging, as they usually have a complex graphical user interface and advanced functionality that involves interacting with remote services. In the context of higher education, assignments and exercises on mobile application programming are also non-trivial to evaluate [12]. As a consequence, student assessment in this context usually relies on assignments or projects that are manually checked by teaching assistants for grading [35]. This approach has several limitations, including scaling to large classrooms. In contrast, automated assessment of student submissions has been largely used in computer science education, primarily for grading exercises in introductory programming courses [11]. Significantly less research, instead, has focused on automated grading for more advanced courses, including mobile application programming [6].

This article presents a novel system for automated assessment of Android exercises with cloud-native technologies. Different from the state of the art (Section 2), the proposed solution leverages a mobile app testing framework that is largely used in the industry instead of custom libraries. Moreover, the devised system employs software containers and scales with the availability of resources in a data center, which is essential for massive open online courses. Such a cloud-friendly system is considered in the context of a course on full-stack development (Section 3).

In particular, this work establishes the following major contributions. First, it introduces **the design and implementation of a system** for automated assessment of Android exercises (Section 4). The proposed solution leverages modern cloud-native technologies and mobile app testing practices widely used in the industry for grading purposes. Second, it **analyzes the impact of the developed system** on learning in a master-level course with 120 students, supported by both behavioral data and student feedback (Section 5). Such analysis demonstrates that the proposed solution was effective, as it provided insightful feedback and supported independent learning of mobile application development. Finally, it **discusses the experience** of employing the proposed system

ITiCSE '20, June 15–19, 2020, Trondheim, Norway

ACM ISBN 978-1-4503-6874-2/20/06.
https://doi.org/10.1145/3341525.3387430

at a university (Section 6). Specifically, it offers important practical considerations on automation for teaching mobile application development in higher education and beyond.

2 RELATED WORK

Several works in the literature targeted teaching mobile computing and (or) application development in computer science, particularly, for novice programmers. Among them, Reardon and Tangney [31] showed the feasibility of employing project-based learning of the Java programming language in the context of mobile application development for freshmen. Ilinkin [18] developed a Java-based framework to facilitate learning of mobile application development in a CS1 course. Black [8] summarized experiences in teaching mobile application programming for Android in a CS2 course, with focus on game development. However, these works are based on courses that leverage tutorials and staff or peer feedback instead of automated tools. In contrast, this article explicitly considers online learning activities through automated assessment.

Automated assessment of student submissions has received wide consideration in the literature [5, 13, 33]; the following focuses on the works that are most relevant. Allevato et al. [6] designed and implemented a system for automated assessment of Android applications. Specifically, they leveraged the Roboelectric framework to run applications without the need for the Android emulator, with some loss in fidelity. Instead, this article demonstrates the feasibility of accurate assessment by applying best practices in software development and cloud-native technologies. Wünsche et al. [36] developed a system for automated grading of computer graphics assignments based on OpenGL. The authors addressed the design of both the system and the exercises used in a course. Their proposed solution was realized as a Moodle plug-in that runs student code within a virtual machine. Even though the context is different, this article shares similarities with [36] in terms of how to evaluate graphical output of student submissions and the use of cloud technologies. However, the solution proposed here employs software containers instead of virtual machines and targets the more complex scenario represented by Android applications.

There are also a few works that specifically considered the use of modern cloud-based technologies in the context of computer science education. JupyterHub [26] leverages software containers to provide computing sessions in the form of Python notebooks. nbgrader [9] extends JupyterHub with support for automated grading of exercises. However, these solutions are specific to interactive labs, thus, more suitable for code snippets rather than for more complex code, such as that for mobile applications. Peveler et al. [29] compared the performance of systems for automatic grading when running into jailed sandboxes and containers. They found that containers are more flexible but incur in some overhead, generally limited to a dozen seconds or less per submission. Unfortunately, their study is restricted to a performance analysis and does not provide any considerations on student learning, as instead done in this article. Maicus et al. [23] described a container-based system for automated assessment of student submissions in a distributed algorithms course. However, their evaluation considered the interactions between the different components as message streams, as opposed to the UI-driven testing addressed here.

3 BACKGROUND

This section introduces first the relevant cloud technologies, then provides the context by describing the university course in which the devised system was deployed.

3.1 Cloud-native technologies

Cloud-native is a term that broadly encompasses both the technical and process-related aspects to build and run applications in a modern cloud computing environment [1, 27]. The key enabling technology in this context is represented by *software containers*, namely, self-contained software images that can easily be packaged and deployed through light-weight (i.e., operating system-level) virtualization [7]. Containers have low startup time and execution overhead [14]; they also allow the realization of distributed applications as microservices [3]. Container *orchestration* involves provisioning, scheduling, and managing containers at scale [2].

Cloud-native application development involves the adoption of agile and continuous integration / deployment practices in software engineering [16]. Accordingly, software development leverages an automatic multi-stage process, triggered by the creation or modification of source code: it not only builds and tests an application or service, but also releases it to the end users as soon as it is ready [22].

3.2 The Mobile Cloud Computing course

The system considered here was devised for course CS-E4100 Mobile Cloud Computing at Aalto University. The course addresses the development of a distributed application for mobile devices by using a cloud infrastructure, including elements of mobile computing. The main goal of the course is to provide students with full-stack development skills, in other words, to enable them to develop both client (i.e., *frontend*) and server (i.e., *backend*) software in the context of modern mobile application development. Accordingly, the course consisted of two main components: mobile application programming with Android for the frontend; and a cloud-based mobile backend. The course is offered in several programs and is primarily intended for first-year master students, even though it accepts undergraduate students. As a consequence, the course assumes some knowledge about programming languages (e.g., Java), client-server programming, as well as data structures and algorithms as a pre-requisite. The course takes place over one semester and has about 120 students per year.

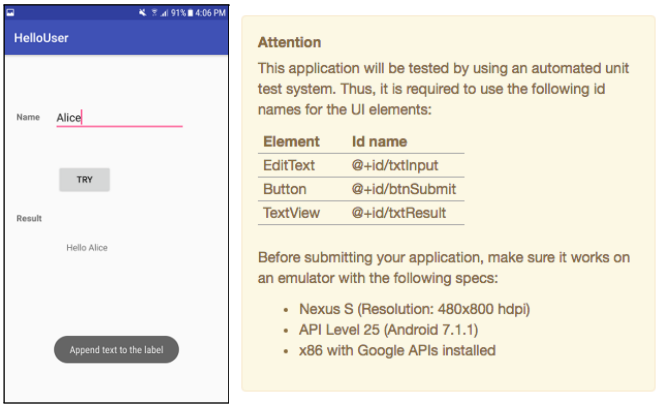
To pass the course, students have to complete individual online exercises submitted through an online platform. They also have to carry out a group project consisting of an Android application and the supporting server components. While the online exercises are automatically graded, a demo of the software project is presented to the teaching assistants.

4 ASSESSING ANDROID EXERCISES

The following introduces first the rationale behind the design of the Android exercises. It then details the implementation of the components that perform automated assessment of these exercises.

4.1 Exercise design

The course included four Android exercises with different level of complexity, corresponding to a varying number of points. All



(a) (b)

Figure 1: Instructions for the Hello User exercise: (a) sample output and (b) reference IDs of UI elements.

exercises are meant to implement some functionalities needed in the realization of the group project. This approach makes it easier for students to get started with the project work, as some of the software components developed for the online exercises could be reused for the project. In detail, the exercises are as follows.

Hello User Given a string as input, the mobile app has to display "Hello" followed by a space and the input string.

QR Code Given a string as input, the mobile app has to display the corresponding quick response (QR) code on the screen.

Image List Given a Javascript Object Notation (JSON) file as input, the mobile app has to display a sequence of items. In particular, each item should consist of an image and of the corresponding author's name, as specified in the input file.

Image Detection Given a source picture, the mobile app has to tell if it contains faces and (or) a barcode. The user interface should include an option to select the image from the gallery of the phone by clicking a button. The functions for face detection and barcode recognition should be implemented by using the Google Mobile Vision API.

Exercises are graded by means of mobile app UI testing based on how the requested functionality is fulfilled. Let us consider the Hello User exercise as an illustrative example: given a string as input (e.g., "Alice"), the mobile app should display "Hello" followed by the given string (e.g., "Hello Alice"). The input is taken from a text box and the result is shown on a text label once the "Try" button is clicked. Figure 1a shows a sample output that is also provided to the students as part of the online material. To ease application testing at the grader, students are asked to use specific ID names for the elements of the user interface, similar to [6], as shown in Figure 1b. This approach follows best practices in mobile app development, used in production for automated UI testing [12].

4.2 Android grader

The Android grader is the component that performs the actual assessment of the online exercises. The grader is a Docker container that includes all software needed to execute and evaluate student submissions. Assessment is carried out by running exercise-specific unit tests on the Android app submitted in binary format, namely, as an Android application package (APK). The rationale behind this choice is that students should build the Android application on their local development host. They should also carefully evaluate the build output and run the application before submitting it online, to check whether it meets the specifications of the exercise or not.

Android exercises are assessed through Appium, an open-source test automation framework[1] primarily targeted for testing mobile / web applications. In particular, Appium is widely used for testing native mobile apps [34], including those for Android. One of the key strengths of Appium is its flexibility: it does not require any modifications to the source app and supports multiple languages as well as platforms. Appium has a client-server architecture: the server offers an Application Programming Interface (API) to execute commands on a mobile device and obtain the corresponding results; the client implements the actual tests by performing actions through the UI elements in the mobile app.

The grader includes Appium, the Java and Android software development kits (SDKs) as well as the related SDK / build tools, and Maven for unit testing. The container also provides the Android emulator through a custom image that has the Google Mobile Vision API libraries pre-installed.

4.3 Exercise feedback

Giving constructive feedback is necessary for students to understand the outcome of their exercise submission, especially when failures occur [20, 28]. Providing insightful feedback on mobile application development is indeed very challenging, as opposed to console-based programs that only have textual output [6].

For this purpose, the exercise feedback offers information about the submission outcome at different levels (Figure 2). The "Test Results" section summarizes individual unit tests (i.e., through their name and description) and the related outcome. Such an outcome reports if the test was completed successfully, providing more specific information in case of errors. In particular, the feedback describes the nature of the failure in an expressive form and gives hints on possible reasons [21]. The grader also provides detailed debugging information. The "Android Logcat Output" section shows custom debugging messages written through the logging API built into Android (i.e., Logcat) and marked with a custom tag. This offers the same level of debugging output generally available for console-based applications. In case of errors, the feedback also shows the stack trace corresponding to the specific instruction that triggered a failure. Finally, a screenshot of the application is also provided to give additional feedback at a glance. Screenshots are taken periodically and the last one stored just before the failure is provided as the submission result.

Hints were compiled based on common mistakes made by student on the basis of prior teaching experience. One of them is displaying user elements in a way that are not fully contained in the view of the device, for instance, due to overflowing. One example is represented by a QR code not completely visible in the related exercise, as shown in Figure 2. The related exercise submission failed because the QR code was not detected. The feedback provided by the system suggests that the problem could be related

[1] `http://appium.io`

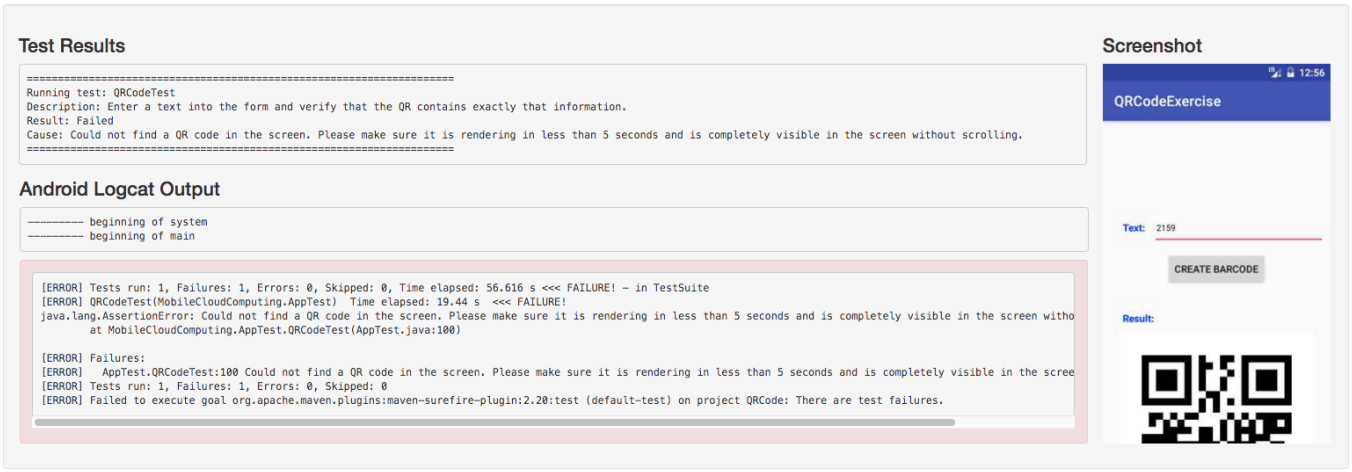


Figure 2: Sample grader feedback for a submission of the QR Code exercise including: a summary of the test results, with a high-level description of the error and hints on possible reasons; debugging information, including log output and stack trace; a screenshot captured right before the test failure.

to the visibility of the QR code on-screen. A quick inspection of the accompanying screenshot reveals that this is indeed the case.

4.4 Using the grader as a teacher

The developed system offers a simple RESTful API, similar to today's cloud-based services. As a consequence, it could easily be interfaced with different learning management systems (LMSs), particularly, those offering built-in support for running container-based graders [19, 30]. The following considers the A+ LMS[2] which was used in the reference course. In such a case, the teacher only needs to: write the unit tests for the exercises; and configure A+ to use the Android grading container. These steps are detailed next.

Unit tests are written by leveraging the Appium client library, which is available in a variety of language bindings. Listing 1 shows an excerpt of the unit test for the Hello User exercise as an illustrative example. The code is written in Java for Maven and uses the Appium API to test the compliance of the submission against the exercise specifications (refer to Section 4.1 for more details). Accordingly, the test first generates a random number (say 42), then inserts its value into the `txtInput` text box and clicks the `btnSubmit` button. Finally, the test extracts the value of the `txtResult` text view and compares it with the expected one ("Hello 42"). The unit test yields the exercise points based on whether the test passes or not. It also returns the hint shown in the "Test Results" section of the submission feedback (see Figure 2).

A certain unit test then needs to be associated with a specific exercise. This can be accomplished, for instance, by adding the unit test to the git repository linked to the course and editing exercise-specific configuration files. These files also specify the name of the software container corresponding to the grader. Finally, the exercise needs to be configured to award a certain number of points and to accept submissions as binary files.

The software described in this article as well as instructions on how to use it are available online at `http://cloudscape.aalto.fi`.

[2] `https://apluslms.github.io`

```
public class AppTest {

  AndroidDriver driver;

  public void HelloUserTest(){
    String currId = null;
    try {
      // Input random value into EditText
      Random rand = new Random();
      int n = rand.nextInt(25000) + 1;
      currId = "txtInput";
      driver.findElement(By.id(currId)).sendKeys(""+n);
      // Press button
      TouchAction t = new TouchAction(driver);
      currId = "btnSubmit";
      t.tap(driver.findElement(By.id(currId))).perform();
      // Check if input text is appended to Hello
      currId = "txtResult";
      Assert.assertEquals(driver.findElement(By.id(currId)).
        ↪ getText(), "Hello "+n, "The TextView contains the
        ↪  wrong text.");
    } catch (NoSuchElementException e) {
      Assert.fail("Could not find element with ID '"+currId+
        ↪ "'. Please check the assignment instruction and
        ↪ use the provided IDs.");
    } catch (Exception e) {
      Assert.fail(e.getMessage());
    }
  }

}
```

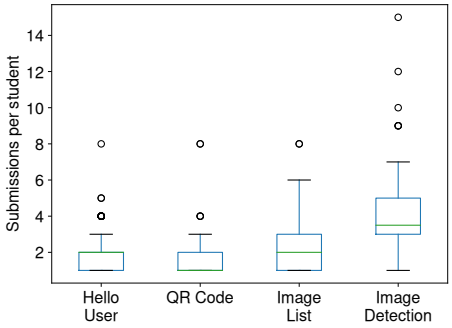
Listing 1: Code of the unit test for the Hello User exercise.

5 EVALUATION

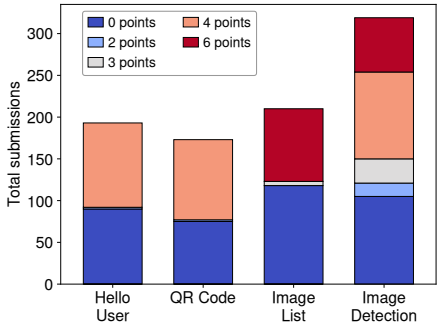
The proposed system is evaluated next in terms of student learning.

5.1 Methodology and setup

The evaluation includes an analysis based on both submission data and a survey. The results refer to a single offering of the course in Fall 2017, with 120 students enrolled at the beginning of the semester – 96 of which completed the course by the end of the year.



(a)



(b)

Parameter	Q1	Q2
Median	4	4
Stdev	0.93	0.91
1 rating	2.17	2.22
2 rating	2.17	6.67
3 rating	17.39	15.56
4 rating	34.78	53.33
5 rating	43.48	22.22

(c)

Figure 3: Submissions of individual Android exercises: (a) average number per student and (b) total number broken down by obtained points. (c) Summary statistics and distribution of ratings in the course feedback questionnaire.

The system was configured to allow at most 20 submissions of a given Android exercise per student. The first two exercises (i.e., Hello User and QR Code) were worth 4 points each, while the other two (i.e., Image List and Image Detection) 6 points each. The Image Detection exercise gave fractional points based on multiple unit tests, while the rest of the exercises awarded either the full amount of points or zero. All Android exercises had the same submission deadline, six weeks after the beginning of the course. Students were allowed to submit exercises up to one week late with a 50% penalty.

The Android grader ran in a private cloud computing infrastructure at Aalto University. In particular, graders were managed and deployed through Kubernetes running on bare metal (i.e., software containers did not run inside virtual machines).

5.2 Exercise submissions

Submission data were analyzed with the pandas data analysis library [25]. The following focuses on the number of submissions by exercise to derive insights on how students used the system and the related impact on learning. There were 895 submissions in total for the four exercises in the considered course offering.

Figure 3a shows the statistical summaries on the number of submissions per student for the different Android exercises. The figure shows that students only turned in a few submissions for the first three exercises, with a median no higher than three and a maximum of at most seven. In particular, the statistical summaries are very close for the two exercises, as they are very simple and quite similar. The number of submissions per student increases as the difficulty of the exercises increases too. The sharper increase for the last exercise is also due to the presence of multiple unit tests which awarded fractional points: students submitted multiple times to obtain full points. The figure clearly shows how the limit of 20 submissions per exercise was effective in discouraging a trial-and-error approach purely based on grader feedback. Finally, there are outliers, which is not unexpected given the class size and the presence of students from different master programs – there were also a few bachelor students enrolled in the course.

Figure 3b shows the total number of submissions for the different Android exercises, broken down by the corresponding number of obtained points. Specifically, the figure illustrates the overall student results in solving the exercises by distinguishing between successful and unsuccessful submissions. The results clearly show how the first two exercises were successfully completed by a large share of the students. The actual numbers for the first two exercises are similar, even though the total submissions is higher for the Hello User exercise. This happened because students had to explore and get familiar with the automated grading system at first. This is supported by the fact that the number of successful submissions is almost the same in the first two exercises, while the number of unsuccessful attempts is lower for the QR code exercise. Nevertheless, the results demonstrate that the overhead in learning how to use the system was quite limited. The figure also shows a different trend for the last two exercises. In fact, the number of submission with full points is lower and the spread in the obtained points higher, which are both consistent with the increasing difficulty of the exercises.

It is possible to relate the learning from the automatically-graded Android assignments to the course project, carried out by groups of five students. The obtained data suggest that the Android exercises helped all groups obtain at least 35% of the project points. Moreover, the overall course completion rate increased by 13% when automated grading was used with respect to the previous year, which included no Android assignments but only two smaller projects.

5.3 Student feedback

Feedback about the system was collected within a course summary questionnaire that was sent to all students enrolled in the course after its completion. A total of 47 students submitted their feedback. The survey included three questions specific to the automated system for assessment of Android exercises: one rating how it supported learning based on a 5-point Likert scale, wherein 1 corresponds to “Strongly disagree” and 5 to “Strongly agree”; and two about the most valuable and least convincing aspects of the system, each allowing unconstrained textual comments. The feedback also included one question about other online material and exercises (unrelated to automated assessment of Android exercises), also rated on a 5-point Likert scale, which is reported as a baseline for comparison purposes. The specific wording for the questions with Likert-scale ratings was the following: “*The automated system used for the evaluation of Android assignments has*

supported my learning" and "*The online material and questionnaires have supported my learning*". These questions are referred to as Q1 and Q2 – respectively – for conciseness in the rest of the article.

Table 3c shows summary statistics on questions Q1 and Q2 as well as the distribution of the corresponding ratings as percentages. The results show how all online activities were very favorably received by the students, with a median rating of 4 and a relatively low standard deviation of about 0.9. However, the actual distribution of ratings reveals that students acknowledged a high impact of the Android assessment on learning in question Q1: about 43.5% gave a rating of 5, almost two times the value obtained for question Q2.

The free-text comments supported such claims, as students even praised the system – "*it was great*", "*I was amazed how well and fast the grading worked*", "*it's actually pretty cool*". Students particularly valued prompt feedback, which was described as "*quick*", "*fast*", and even "*instant*". Students also appreciated its informative value, especially the "*explanation of the errors*". Specifically, the feedback was found "*very informative*"; students explicitly mentioned "*the response[s] are really helpful*", "*error reporting which helped with debugging*", "*screenshots with proper reasoning for the errors if any*". A few students appreciated automation, not only because of the related technical aspects, but also for fairness ("*no one can complain or try to ask better grades because you get what you deserve*"). Some feedback explicitly pointed out the impact on learning: "*it forced me to learn [...] which is a good thing and I am happy that I did it*", "*I [was] always motivated with [the] automated grading system*". Some issues were also pointed out, mostly related to technical problems: "*few bugs in [the] testing tool*", "*some technical hickups and delays in the beginning*". A few students, instead, felt that the system was not fast enough: "*it was a bit slow*", "*the processing speed is low*"; others expressed concerns on cheating [24]: "*I guess one could have kinda cheated by just hardcoding the desirable outputs*", "*one could've pretty easily gotten full points without a proper submission*".

6 DISCUSSION

The following discusses important aspects of the proposed system based on both student feedback and course experience.

Grader performance. Building and running Android applications is a time-consuming process, even when performed in a powerful data center [6]. In this respect, some measures were taken to reduce the time needed to assess student submission. First, students were required to submit the Android application binary (as APK), so that the grader does not have to build the application from source. This also reduces the load on the computing infrastructure as a side effect. Second, the grader employed the x86 version of the emulator, which is much faster than that for ARM [4]. The x86 emulator requires hardware virtualization, which was made available to the container by sharing the corresponding element in the device filesystem. This solution allowed to reduce the time needed for assessing exercises by half: from five minutes per submission to a few.

Software frameworks. The decision on having binary application submissions allowed students more flexibility in choosing software frameworks [32]. The course explicitly referred to programming Android applications in Java, which was the only option supported by the course staff. However, students were allowed to pick other programming languages or software frameworks, with no promise of their compatibility with the system for automated assessment of Android exercises. Interestingly, some students started exploring such alternative options and managed to use them for successful submissions. Kotlin[3] and React Native[4] were the most popular solutions. This also demonstrates the flexibility of proposed solution.

Reliability. The reliability of the system during the course was generally remarkable. A few bugs actually affected the code for assessing the exercises; they were promptly fixed as students reported issues. Interestingly, the main sources of unreliability were due to factors that were not under the control of the course staff. One of them was a service break on the IT services at the university which took place during a weekend right after the beginning of the semester; another was the grading backend that stopped following a crash of the grader, thereby preventing student submissions to be assessed. It was enough to clear the submission queue to make the system operational again.

Effort of teaching staff. The system was realized in less than a semester by a group of five people, including a teacher, two teaching assistants, and two interns. The initial version of the exercises required some effort in handling all corner cases that could happen with a large student audience. Nevertheless, the total effort by the course staff was definitely lower than designing and evaluating individual student projects. As a term of comparison, the Mobile Cloud Computing course included two student projects before automated grading was introduced to achieve the same learning outcomes. Since the initial deployment, the system has effectively been used in other settings – including an online course on Android development open to a broader audience – with very limited effort.

7 CONCLUSION

This article presented a system for automated assessment of Android exercises through the experience gained from its application to a university-level course. The proposed solution leverages modern cloud-friendly technologies and mobile app testing practices widely used in the industry for grading purposes. The system was shown to be effective in terms of student learning based on an analysis of exercise submissions and a student feedback questionnaire. Future research could address how to detect plagiarism in this specific context. Evaluating the proposed solution across courses at different levels – i.e., bachelor as opposed to master, but also in contexts other than higher education – is also an interesting research direction.

ACKNOWLEDGMENTS

This work was partially supported by Aalto University under the CloudScape pilot of the Aalto Online Learning[5] project. The authors would like to thank: Jan-Mikael Rybicki and Lauri Malmi for their feedback on a preliminary version of this article; Manoj Kumar, Prasant Sukumar, Gopika Premsankar, Teemu Lehtinen, Mikko Hakala, Markus Murhu, and Markku Riekkinen for their help in the practicalities needed to arrange the course; the Aalto Science-IT project for provisioning the computational resources used.

[3] https://kotlinlang.org
[4] https://reactnative.dev
[5] https://onlinelearning.aalto.fi

REFERENCES

[1] Cloud Native Computing Foundation. CNCF Cloud Native Definition v1.0. Available online: https://github.com/cncf/toc/blob/master/DEFINITION.md. Accessed January 19, 2020.

[2] Isaac Eldridge. What Is Container Orchestration? Available online: https://blog.newrelic.com/?p=43457. Accessed January 19, 2020.

[3] Martin Fowler. Microservices – a definition of this new architectural term. Available online: https://martinfowler.com/articles/microservices.html. Accessed January 19, 2020.

[4] The Chromium Project. Using an Android Emulator. Available online: https://chromium.googlesource.com/chromium/src/+/HEAD/docs/android_emulator.md/. Accessed January 19, 2020.

[5] Kirsti M. Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15, 2 (2005), 83–102. https://doi.org/10.1080/08993400500150747

[6] Anthony Allevato and Stephen H. Edwards. 2012. RoboLIFT: Engaging CS2 Students with Testable, Automatically Evaluated Android Applications. In *The 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*. 547–552. https://doi.org/10.1145/2157136.2157293

[7] David Bernstein. 2014. Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing* 1, 3 (Sep. 2014), 81–84. https://doi.org/10.1109/MCC.2014.51

[8] Michael David Black. 2016. Seven Semesters of Android Game Programming in CS2. In *The 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. 5–10. https://doi.org/10.1145/2899415.2899470

[9] Douglas S Blank, David Bourgin, Alexander Brown, Matthias Bussonnier, Jonathan Frederic, Brian Granger, Thomas L Griffiths, Jessica Hamrick, Kyle Kelley, M Pacer, et al. 2019. nbgrader: A tool for creating and grading assignments in the Jupyter Notebook. *The Journal of Open Source Education* 2, 11 (2019).

[10] Barry Burd, João Paulo Barros, Chris Johnson, Stan Kurkovsky, Arnold Rosenbloom, and Nikolai Tillman. 2012. Educating for Mobile Computing: Addressing the New Challenges. In *The Final Reports on Innovation and Technology in Computer Science Education 2012 Working Groups (ITiCSE-WGR '12)*. 51–63. https://doi.org/10.1145/2426636.2426641

[11] Julio C. Caiza and José María del Álamo Ramiro. 2013. Programming assignments automatic grading: review of tools and implementations. In *The 7th International Technology, Education and Development Conference (INTED2013)*.

[12] Riccardo Coppola, Maurizio Morisio, and Marco Torchiano. 2019. Mobile GUI Testing Fragility: A Study on Open-Source Android Applications. *IEEE Transactions on Reliability* 68, 1 (2019), 67–90. https://doi.org/10.1109/TR.2018.2869227

[13] Christopher Douce, David Livingstone, and James Orwell. 2005. Automatic Test-Based Assessment of Programming: A Review. *J. Educ. Resour. Comput.* 5, 3, Article 4 (Sep 2005), 13 pages. https://doi.org/10.1145/1163405.1163409

[14] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. 2015. An updated performance comparison of virtual machines and Linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 171–172. https://doi.org/10.1109/ISPASS.2015.7095802

[15] James B. Fenwick, Jr., Barry L. Kurtz, and Joel Hollingsworth. 2011. Teaching Mobile Computing and Developing Software to Support Computer Science Education. In *The 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. 589–594. https://doi.org/10.1145/1953163.1953327

[16] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software* 123 (2017), 176 – 189. https://doi.org/10.1016/j.jss.2015.06.063

[17] Daniel Galan, Ruben Heradio, Hector Vargas, Ismael Abad, and Jose A. Cerrada. 2019. Automated Assessment of Computer Programming Practices: The 8-Years UNED Experience. *IEEE Access* 7 (August 2019), 130113–130119. https://doi.org/10.1109/ACCESS.2019.2938391

[18] Ivaylo Ilinkin. 2014. Opportunities for Android Projects in a CS1 Course. In *The 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. 615–620. https://doi.org/10.1145/2538862.2538983

[19] Ville Karavirta, Petri Ihantola, and Teemu Koskinen. 2013. Service-Oriented Approach to Improve Interoperability of E-Learning Systems. In *2013 IEEE 13th International Conference on Advanced Learning Technologies*. 341–345. https://doi.org/10.1109/ICALT.2013.105

[20] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2016. Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. In *The 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. 41–46. https://doi.org/10.1145/2899415.2899422

[21] Tobias Kohn. 2019. The Error Behind The Message: Finding the Cause of Error Messages in Python. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. 524–530. https://doi.org/10.1145/3287324.3287381

[22] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A Survey of DevOps Concepts and Challenges. *ACM Comput. Surv.* 52, 6, Article 127 (Nov. 2019), 35 pages. https://doi.org/10.1145/3359981

[23] Evan Maicus, Matthew Peveler, Stacy Patterson, and Barbara Cutler. 2019. Autograding Distributed Algorithms in Networked Containers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. 133–138. https://doi.org/10.1145/3287324.3287505

[24] Tony Mason, Ada Gavrilovska, and David A. Joyner. 2019. Collaboration Versus Cheating: Reducing Code Plagiarism in an Online MS Computer Science Program. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. 1004–1010. https://doi.org/10.1145/3287324.3287443

[25] Wes McKinney. 2011. pandas: a foundational Python library for data analysis and statistics. *Python for High Performance and Scientific Computing* 14 (2011).

[26] Michael Milligan. 2017. Interactive HPC Gateways with Jupyter and Jupyterhub. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact (PEARC17)*. Article 63, 4 pages. https://doi.org/10.1145/3093338.3104159

[27] Claus Pahl, Pooyan Jamshidi, and Olaf Zimmermann. 2018. Architectural Principles for Cloud Software. *ACM Trans. Internet Technol.* 18, 2, Article 17 (Feb. 2018), 23 pages. https://doi.org/10.1145/3104028

[28] Lilian Passos Scatalon, Ellen Francine Barbosa, and Rogerio Eduardo Garcia. 2017. Challenges to integrate software testing into introductory programming courses. In *2017 IEEE Frontiers in Education Conference (FIE)*. 1–9. https://doi.org/10.1109/FIE.2017.8190557

[29] Matthew Peveler, Evan Maicus, and Barbara Cutler. 2019. Comparing Jailed Sandboxes vs Containers Within an Autograding System. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. 139–145. https://doi.org/10.1145/3287324.3287507

[30] Matthew Peveler, Jeramey Tyler, Samuel Breese, Barbara Cutler, and Ana Milanova. 2017. Submitty: An Open Source, Highly-Configurable Platform for Grading of Programming Assignments. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 641. https://doi.org/10.1145/3017680.3022384

[31] Susan Reardon and Brendan Tangney. 2014. Smartphones, Studio-Based Learning, and Scaffolding: Helping Novices Learn to Program. *ACM Trans. Comput. Educ.* 14, 4, Article 23 (Dec. 2014), 15 pages. https://doi.org/10.1145/2677089

[32] Amit Shesh. 2019. Allowing and Fully Supporting Multiple Programming Languages in a Computer Graphics Course: An Experience. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. 239–245. https://doi.org/10.1145/3287324.3287464

[33] Draylson M. Souza, Katia R. Felizardo, and Ellen. F. Barbosa. 2016. A Systematic Literature Review of Assessment Tools for Programming Assignments. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*. 147–156. https://doi.org/10.1109/CSEET.2016.48

[34] Oleksii Starov, Sergiy Vilkomir, Anatoliy Gorbenko, and Vyacheslav Kharchenko. 2015. Testing-as-a-Service for Mobile Applications: State-of-the-Art Survey. In *Dependability Problems of Complex Information Systems*. 55–71. https://doi.org/10.1007/978-3-319-08964-5_4

[35] Kelvin Sung and Arjmand Samuel. 2014. Mobile Application Development Classes for the Mobile Era. In *The 2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)*. 141–146. https://doi.org/10.1145/2591708.2591710

[36] Burkhard C. Wünsche, Zhen Chen, Lindsay Shaw, Thomas Suselo, Kai-Cheung Leung, Davis Dimalen, Wannes van der Mark, Andrew Luxton-Reilly, and Richard Lobb. 2018. Automatic Assessment of OpenGL Computer Graphics Assignments. In *The 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)*. 81–86. https://doi.org/10.1145/3197091.3197112