A Script-Based Approach for Teaching and Assessing Android Application Development

PAOLO MODESTI, Department of Computing and Games, Teesside University, UK

Mobile applications are extremely popular with many higher education institutions offering courses to prepare new developers sought by the software industry. However, teaching and assessing mobile application development poses specific challenges due to the complexity of real-world programming languages and environments. In this work, we present a script-based approach for teaching and assessing Android application development which addresses shortcomings of existing tools that impact negatively on the learning experience. Our evaluation, that covers pedagogical and technical aspects, provides possible evidence that the scripts have been beneficial in helping students to work more efficiently and achieve better results. Additionally, the scripts have been effective in streamlining the grading process and keeping the tutorial material up to date with the evolution of the Android platform.

CCS Concepts: • **Software and its engineering** → **Software development methods**; *Command and control languages*; • **Human-centered computing** → *Smartphones*.

Additional Key Words and Phrases: Android, mobile application development, teaching and assessment, semi-automatic grading, command-line scripting

ACM Reference Format:
Paolo Modesti. 2021. A Script-Based Approach for Teaching and Assessing Android Application Development . *ACM Trans. Comput. Educ.* 21, 1, Article 7 (January 2021), 30 pages. https://doi.org/10.1145/3427593

1 INTRODUCTION

Mobile applications are extremely popular and developers in this sector are highly requested by the software industry. In order to provide students with the knowledge and skills required to become proficient mobile developers, higher education institutions have introduced specialized courses in their curricula.

However, instructors face specific challenges, not only because the design and implementation of mobile applications is a complex task [21, 53], but also because real-world development tools can make the learning curve steeper. Despite that, since exposure to real-world tools used by the industry is a key learning objective in

Author's address: Paolo Modesti, p.modesti@tees.ac.uk, Department of Computing and Games, Teesside University, Middlesbrough, UK, TS1 3BX.

1946-6226/2021/1-ART7 $15.00
https://doi.org/10.1145/3427593

this subject area [44], the choice of languages and tools in academic courses is often driven more by the employability needs than by the pedagogical ones.

In the mobile domain, Google's Android is the most popular operating system (OS) with a market share of 86% [30]. According to Google, there are currently more than 2.5 billion monthly active Android devices [2]. Moreover, Android Studio [5], made by Google and powered by IntelliJ, is the official Integrated Development Environment (IDE) for Android application development. For this reason, Android Studio has been adopted in many courses, including the "Android Mobile Development" course that the author has delivered at the University of Sunderland (UK) in the period 2016-18. This is a predominantly programming oriented module where students learn how to design and develop native mobile applications, focusing on Java programming for the Android platform.

Contribution. In this work, we present a script-based approach for teaching and assessing Android application development which addresses technical shortcoming of existing tools which impact negatively on the learning experience. A set of command-line scripts[1], written by the author to complement the Android Studio IDE, was developed with the aim to help students to perform in a simpler and faster manner the typical tasks carried out in a software development session (e.g. clean, build, run, test), manage Android projects, configure the development environment and the virtual/physical devices where these applications run. Devised initially to solve a practical problem (the slow performance of Android Studio), these scripts have been further developed and have become an integral part of the teaching strategy employed in the course.

The evaluation is aimed at understanding if these scripts, also intended as supporting technologies for learning and assessment, have been beneficial in helping students to work more efficiently and produce a higher quality work (i.e. achieving better grades w.r.t. the learning objectives and the assignment requirements). Along with the reflection on the teaching practice, this evaluation is based on:

- data collected from students by means of a questionnaire;
- data analysis of the course assessment;
- analysis of software artefacts submitted by students.

The students' feedback and the results of the evaluation show that the scripts have been useful to work with Android projects more efficiently, and there is possible evidence that students who used the scripts generally performed better than those who did not.

Additionally, an extended set of scripts was also designed to support the marking process, allowing the marker to streamline the activities, saving a considerable amount of time otherwise spent to perform mundane operations, preliminary to the marking itself. The scripts can also be used to automatically upgrade to new versions of Android and validate the demonstration apps and exercises used in lectures and

[1]Available at https://paolo.science/android/

tutorials. In fact, the dynamic nature of this application domain requires frequent changes to adapt the teaching material to the evolution of the Android platform.

Outline of the paper. In §2, we discuss challenges, approaches, and tools for teaching Android application development. In §3, we focus on the issues related to the practical delivery and present our scripts-based approach. §4 covers the evaluation of the developer's scripts and in §5 the evaluation of the admin script's, including the marking process. We conclude in §6, summarising the results and discussing future research directions.

2 TEACHING ANDROID APPLICATION DEVELOPMENT

Given the wide popularity of mobile devices, universities have added to their curricula a significant number of courses on native mobile development. Higher education instructors teaching in computer science courses [20, 28, 32, 44, 50, 61] have highlighted the following challenges:

- the fairly sophisticated skills required to develop native mobile applications and the steep learning curve;
- the difficulties to separate teaching principles from the technical details of the platform;
- the fast rate at which mobile platforms are changing;
- the overall complexity (and some shortcoming) of the developments tools.

However, there are also opportunities such as:

- the effectiveness of mobile platforms in teaching programming;
- the great variety of concepts and techniques that can be taught;
- the valuable experience, also in terms of employability, of working with real-world systems;
- the possibilities to engage students.

2.1 Learning Programming

In general, learning programming is considered to be difficult. Edsger Dijkstra in a seminal paper with the striking but realistic title "*On the cruelty of really teaching computing science*" [15] argued that programming is a "radical novelty" in which the process, typical of many learning systems, of transforming the "novel into the familiar" no longer works.

Jenkins [33] believes that a crucial reason for this is that programming is a skill and not a body of knowledge and there is a general consensus [6, 33, 54, 57] on the fact that deep learning [43] is crucial for software developers.

Ben-Ari [6] indicates that, for some aspects, programming is also a "pattern matching" process, an activity which can be associated with surface learning. In fact, programmers can apply known solutions to common problems to solve a new one. Surface learning can be useful to remember syntactical details but deep learning is necessary to develop a real competence in writing programs. According to Jenkins [33], the crucial point is that the two learning styles must be applied at the same time. In practical terms, teachers of programming courses face several pedagogic and technical choices:

- teaching style: principle vs tutorial-based, need to adapt to students learning with different styles and paces [51, 54];
- programming language: teaching a language used in the industry or a language suitable for its pedagogical features [1, 41, 54];
- tools used in the practical sessions (compilers, debuggers, IDEs, etc.) [24, 41, 52];
- assessment and grading strategy [46].

We consider the first three issues in this section, and we cover the last one in §4 and §5.

2.2 Programming for Android

Although this article focuses on Android, most observations will also apply to iOS and other modern mobile platforms. In general, according to several authors [21, 44, 53], designing and implementing mobile applications is a complex task because the developer has to master a wide range of tools [25, 27]:

- programming languages (e.g. Java, Kotlin, C++, Objective-C, Swift);
- operating systems (e.g. Android, iOS);
- development tools (e.g. Android Studio, Eclipse, XCode).

Developing applications for Android requires a substantial programming background [32] which includes proficiency in Object-Oriented Programming (OOP), event driven programming and Android architecture (e.g. activity life-cycle of an application, app components).

2.2.1 Learning Challenges. Matos and Grasser [44] believe that the complexities in developing mobile applications are of an order of magnitude more challenging than a desktop application. Riley [53] considers the learning curve steep enough to keep most students away from learning Android development on their own. He also underlines the amount of time instructors must spend in dealing with specific technical issues of the Android platform. According to Burd et al. [9], this is unavoidable, because in this domain it is difficult to separate the theoretical principles from the realities of the computing environment. However, Sprinkle [56] recommends teaching mainly general topics, and believes that students must learn on their own, from the documentation, the API of the platform focusing only on the topics relevant to their tasks.

2.2.2 Android as a Platform for Teaching Programming. Despite these challenges, Android is considered to be an effective platform for teaching Java programming as it offers the opportunity to learn a wide range of computer science concepts and techniques [44, 53]. Students engagement can be strong, as they value the first-hand experience with real systems and the professional skills that can be gained [9, 20, 44]. These skills can support their employability and, interestingly, because of their assignment efforts, students may be able to produce useful apps that are potentially marketable. In this regard, choosing Android over iOS, reduces the entry barrier for the distribution through the official app store [36].

2.3 Android Development Tools

The Android Software Development Kit (SDK) is a set of tools used to develop applications for the Android platform. It includes software libraries, debuggers, profilers, device emulators, the API documentation, and code samples. At every new release of the Android OS, an updated version of the SDK is also made available. In little more than a decade, Google has released 11 major revisions of the OS and 30 different levels of the API. Since 2015, Google has developed Android Studio which has become the official IDE for Android development, superseding Eclipse.

2.3.1 IDE Adoption. Fuchs et al. [23] argue that adopting IDEs and tools for teaching software development is a challenge for educators. The adoption of complex IDEs like Android Studio and Eclipse allows students to work with tools used in the industry but makes the learning curve steeper. From the instructor's point of view, this requires more supervision as professional IDEs are very complex. On the one hand, there are so many options that the learner can be overwhelmed and find it difficult to focus on specific learning objectives. On the other hand, the exposure to real-world tools is a learning priority for this kind of subject, as employers require knowledge of specific development tools and programming languages.

However, the richness of functionalities in such tools comes at a price of the required hardware resources. Therefore, their performance may be sluggish even on mid-range machines (for example in university labs), and students may experience difficulties on their own machines as well. This can be a source of frustration when building, testing, and debugging applications. Additionally, apps are often tested on virtual devices and this can make the overall process slower, taking time and energy that should be used for the programming task [32].

2.3.2 Platform Evolution. The pace at which the Android landscape evolves to support new services, sensors, and devices (phones, tablets, smart watches, smart TVs, etc.) is another significant issue. This poses a challenge to instructors because teaching material and labs activities must be kept up to date to adapt to the evolution of the platform [53]. The API changes are too fast for developers and this has implications on the quality of the code (e.g. software defects, security issues) [47]. It may also affect the compatibility of the code across different API levels [29]. This can be a source of confusion for students, as books and online resources they consult can be outdated. These problems also affect the stability of the IDE and other development tools. For example, 11 major and 17 minor releases of Android Studio were made available in four years, from April 2016 (2.0.0) to April 2020 (3.6.3). Worryingly, even a major release like the 2.3.0 was overtly presented as "primarily a bug fix and stability release" in the official release notes [5].

3 SCRIPT-BASED APPROACH

Since Android Studio is the de facto IDE for native Android development, it was the natural choice for our course. Unfortunately, we experienced the same issues described in §2.3.

3.1 Practical Issues

Despite the lab machines meeting the recommended hardware specifications, the students' experience with this Android Studio under Windows was quite frustrating. Activities like cleaning and building a project, based on the Gradle build system [26], were slow and time consuming. The sluggishness of Android Studio/Gradle in the labs (where users do not have admin rights) was a significant grievance for students when they began the course. They also faced the same issues on their own PCs.

This was not a confined problem because the course was due to be delivered by local tutors in the next semester as part of a transnational education (TNE) programme, in a developing country like Botswana where the availability of adequate hardware resources is scarcer than in our campus.

3.1.1 CLI and Alternative Options. Since the Android SDK includes command-line tools, a possible alternative is trying to use them. A significant number of actions, including running Gradle, can be performed faster with the Command Line Interface (CLI) than with the Android Studio's Graphical User Interface (GUI). This can be explained by the fact that, along with the resources used by the IDE itself, there is no overhead of interaction between Android Studio and the underlining tools (e.g. Gradle, Android Debug Bridge).

It should be noted that, despite the advantages of the CLI (simplicity, speed and performance) most users tend to prefer GUIs for their ease, interactivity and greater control [58]. In our case, it also turned out (see §4, Table 1) that the large majority of the students (78%) were not aware of the possibility to build Android applications from the command line.

In order to allow students to work more efficiently, we decided to develop a set of scripts to support the trainees in many activities, such as building, cleaning and running projects, and the configuration of the environment (§3.3.1). We could have just directed the students to the documentation on how to run the command-line tools available in the Android SDK, but it would have added further complexity to the material already taught. Therefore, the scripts allowed the students to focus on learning the core Android design and programming techniques for their projects. Moreover, coding scripts is also a good engineering practice, as the scripts allow running commands without the need to type long lists of parameters, combine sequences of commands and use constructs like selection and iteration to define workflows. Moreover, as the source code of the scripts was made available, students were offered the opportunity to study, adapt and improve them.

In principle, many of the environment configuration issues could have been handled by supplying students with a VM pre-configured with the IDE and supporting tools. However, this solution would have required even more powerful hardware than available at the time and, in general, it would have added further slowness due to virtualisation. Moreover, we need to consider compatibility/stability issues due to the nested virtualisation (i.e. running the Android VM within the virtualised environment). For example, nested virtualisation, limited to the most recent hardware, has been supported by the popular free and open source virtualisation software

Virtual Box only since version 6.1, released in December 2019. Unfortunately, in our experience such support is not very stable yet, leading to occasional machine freeze or corruption.

3.1.2 Cross-Machine Issues. Another specific challenge is that, in order to run and test an app, it is necessary to build it on the developer machine and then load it on a physical or a virtual device, like the Android Virtual Device (AVD) emulator. While this is not a major issue in the production environment, where the configuration of the tools is under the developer's control, in educational settings applications often run on systems on which the instructor has little or no control. Typically, the instructor prepares examples which are made available to be studied, edited, run and tested during the practical session in the labs, where the instructor is still in control of the environment. But the examples must also run on students' machines, where such control no longer exists.

In Android Studio, building an app developed on another system implies importing the source code in the IDE. This may also require a certain amount of refactoring depending on the version of the Android SDK and API level of the source and target platform. Given the variety of configurations, the import function of the IDE is critical and far from being faultless. Cleaning and building a project, even a fairly simple one, can take several minutes. Additionally, projects will not compile if the required versions of tools/libraries are not present in the system. This is critical in labs where students do not have admin rights, and cannot freely install new components or versions of the SDK. Therefore, when exercises and assignments are submitted, if students do not align with the expected configuration, the tutor will be likely to face the same problem.

Halper [28] remarks that "the logistics of having to import all the code and run the apps with sample data is very time-consuming". The variety of possible configurations has also been identified as a critical issue. The same author also stresses that even "the rather mundane issue of student project submission and testing can be a major headache for a course instructor". For example, Madeja and Porubän [42] report that most Android projects submitted by students needed some manual intervention.

To avoid this burden, students are often asked to demonstrate their apps to the instructor. We think that this is not ideal as it limits the possibility to fully evaluate different aspects of the work, making the assessment partial at best. It is also frustrating because traditional programming assignments can be assessed in batch mode using scripts that can automatically run and test programs [31].

3.2 Areas of Intervention

To addresses these problems, we considered developing a set of scripts designed with the following objectives:

- speed up operations related to the development process (e.g. clean, build, run on AVD);
- minimize the risk of incompatibility when converting or moving a project to other systems or platform versions;

- allow to check the alignment of the development environment or a project w.r.t. a specific ("target") configuration;
- facilitate the submission of exercises and assignment work.

It should be noted that, according to the taxonomy identified by Pascarella et al. [49], these objectives cover significant aspects of three (out of nine) self-reported activities areas for Android developers. Additionally, from the tutor's point of view, we considered how to:

- minimize the resources required to demonstrate apps during lectures/tutorials;
- control the configuration of the teaching material and automate the migration to new versions of the API/SDK/IDE;
- make the testing and marking process more efficient.

The last issue is crucial as a lot of time is usually spent by tutors to test and mark the students' work, especially when cohort numbers are large. The idea is that, provided that the students submit projects passing a compliance test (e.g. project configuration and sanity checks), the marker can setup an environment that allows to clean and build all students submissions in one batch. Running in a batch mode, instead the using the IDE, has the advantage of not requiring the marker's interaction, saving time and effort. At any time, the marker can inspect the logs and see the progress, checking if any project fails to be compiled. If a compile error occurs at this stage, the marker has the option to contact the student and ask them to fix/discuss the issue and/or use this information as part of the overall assessment.

Although most of the core marking must still be done manually (the marker needs to run each project individually, and grade it according to a marking scheme), a significant amount of time can be saved with the procedure described in §3.3.2, avoiding a large number of tedious repetitive manual interactions with the interface of the IDE. Such procedure automatically installs and runs the app in the AVD, runs the unit tests on the Java Virtual Machine (JVM) and the instrumented tests on the AVD, displays the output of the tests, opens with an editor the relevant source code files that the marker needs to inspect.

3.3 Technical Solution: Command-line Scripts

The technical solution we propose consists of two subsets of command-line scripts, one intended for the *developer* (student) and the other one for the *admin* (tutor). These scripts are batch files for Windows, as this was the OS available in the university labs. In principle, these scripts can be adapted and ported to other platforms including Linux and MacOS.

It should be noted that, from the developer's point of view, these scripts should not be seen as an alternative to Android Studio, but rather as complementary tools to simplify and make specific procedures more efficient. Therefore, when appropriate, the scripts can be used in parallel with Android Studio, just running them from the CLI, rather than using the equivalent functions available through the IDE menus.

Developing Android applications purely from the CLI, without Android Studio, would imply losing access to interactive GUI functions that allow automatic refactoring of code and resource identifiers. For example, the `R.java` file is automatically created and updated by Android Studio at any change made in the XML resource files. Furthermore, while in the past the Android SDK allowed to create a project and a bare-bone application, from the command line (`android create project`), this option has been removed since Android SDK tools version 26.

3.3.1 Developer's scripts. This subset of scripts is aimed at facilitating and making the management of Android projects more efficient along with the configuration of the development and runtime environment.

Given `<p>`, the path of the folder containing an Android project, the following commands apply to a single Android project:

- `clean_single <p>` – Clean the project
- `build_single <p>` – Build the project
- `run_single <p>` – Build and run the project on a device, start the emulator if necessary
- `run_single_apk <f>` – Run an Android package file (APK) on a device, where `<f>` is the filename
- `zip_single <p>` – Clean and zip the project (useful to submit, send or move a project to another machine)
- `test_single <p>` – Run the tests defined in the project
- `setSDK_single <p>` – Reset the SDK path in the project, useful when a project is moved to another system

The commands `clean`, `build`, `zip`, `setSDK` perform the same tasks as above but they process all the projects in a folder. Other commands are available to configure, and control the environment:

- `config` – Define parameters used by the scripts (called by other scripts)
- `config_X.Y.Z` – Define parameters for a specific version of Android Studio/SDK, where `X.Y.Z` is the version number (e.g `2.3.3`) of Android Studio
- `setenvSDK` – Set the system environment variables for the Android SDK (`ANDROID_HOME`, `ANDROID_SDK_HOME`, `GRADLE_HOME`)
- `emulator_start` – Start the AVD emulator if a physical device is not connected to the developer's machine
- `emulator_wait` – Wait the emulator to be up and running (called by other scripts, not invoked directly)
- `packages` – List folders and package names of projects already built
- `projects <p>` – List folders containing Android projects in path `<p>`
- `findSDK` – Find the SDK location (checking a list of possible folders)
- `chkSDK` – Check the current configuration; the optional parameter `<v>` prints the list of all installed and available packages
- `list_devices` – List all attached devices (virtual and physical)
- `kill_device <device>` – Kill an attached device

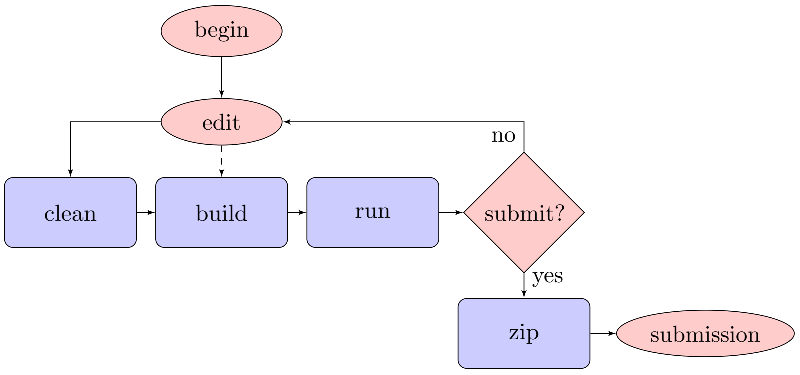

Fig. 1. Developer's workflow

Provided that the environment is configured correctly (`config`, `setenvSDK`, `chkSDK`), a typical session of a student's work (Figure 1), along with the code editing, will consist of cleaning (`clean_single`), building (`build_single`) and running a project (`run_single`). If the project needs to be submitted (or moved to another machine), a compressed archive can be prepared with the `zip_single` command, which includes the cleaning process.

3.3.2 Admin's scripts. This subset of scripts is aimed at supporting and facilitating the tutor in the preparation of demo applications, exercises and the marking process.

Given `<p>`, the path of the folder containing an Android project, the following commands apply to a single project:

- `refactor_single <p>` – Refactor a single project (reorganise files inside a project folder to make them suitable to be used by the scripts)
- `mark_single <p>` – Mark a single project (build, install, run, test, uninstall, display the student's report and source code)

Given `<f>`, the path of the folder containing Android projects, the following commands apply to all projects inside `<f>`:

- `refactor <f>` – Refactor projects, preparing them for marking
- `migrate <f>` – Align projects to a specified "target" configuration (e.g. API level, build tools version)
- `prepare <f>` – Refactor + migrate + clean
- `sanity <f>` – Run sanity checks on projects (check for presence/location of files/folders specified by the tutor within the projects, e.g. build folder)

Provided that the environment is configured correctly, a typical session of the tutor/marker (Figure 2) consists in collecting the students' submissions in a folder `<f>`. Then the projects need to be prepared for marking (`refactor <f>`, `sanity <f>, migrate <f>`, `clean <f>`, `build <f>`) and graded individually (`mark_single <p>` for each `<p>` in `<f>`). The details of the marking procedure are presented in §5.

It should be noted, that there is no need to use Android Studio in this phase, but the only requirement is having the Android SDK installed.

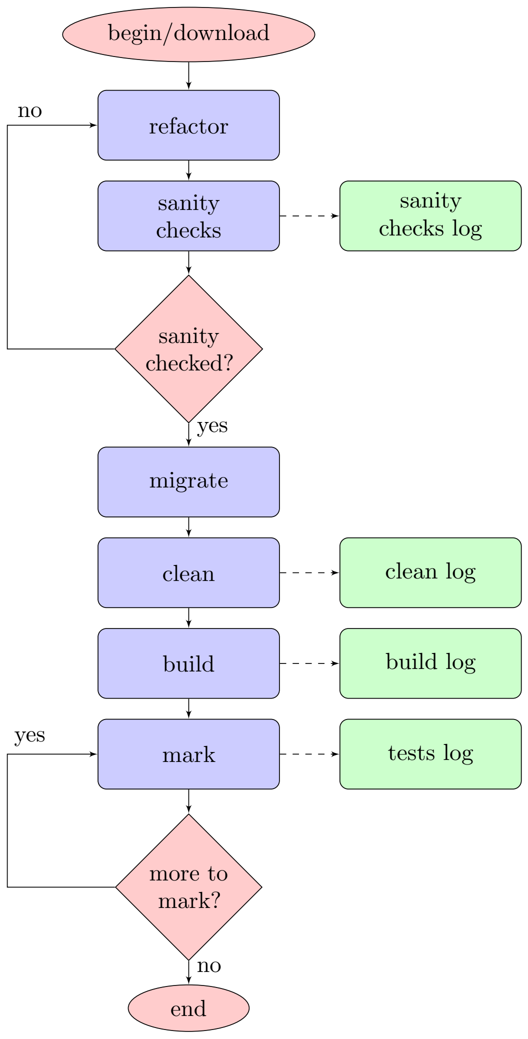

Fig. 2. Admin's workflow (for details about marking/test log see Figure 4)

4 EVALUATION OF THE DEVELOPER'S SCRIPTS

The evaluation is aimed at understanding the impact of these scripts, in particular if they have been useful to help the students to work more efficiently, learning more effectively and achieving a higher standard of work. Along with the reflection of the teaching practice, this evaluation is based on:

- data collected from students by means of a questionnaire (§4.2);
- data analysis of the course assessment and analysis of software artefacts submitted by students (§4.4).

We collected the evaluation data in two consecutive academic years, with two different cohorts of students.

The evaluation methodology is inspired by the first three levels of the Kirkpatrick model [35] for assessing training programs. Chatzigeorgiou et al. [11] used this approach to evaluate an entire Android development course, but here we mostly focus on the tools used in the training sessions.

Firstly, we consider the "*reaction*" of the students, i.e. how they used and assessed the scripts. We also collected information about their prior knowledge of the topic. Secondly, we look at the "*learning*" aspect, i.e. how much has been learned. For this level, we consider a summative assessment to evaluate the extent by which the learning outcomes have been achieved, in terms of students' performance. Therefore, we examined the data from the module results. Thirdly, we considered the "*behaviour*" to understand what changes occurred in the learner attitude and interest for Android development and tools. For practical reasons, data for the "*reaction*" and "*behaviour*" levels have been collected with a single questionnaire.

4.1 Assessment Strategy

Students were asked to develop a native Android application and the requirements specification played a crucial role in the assessment strategy. On the one hand, in agreement with Burd et al. [9], we wanted to give students a concrete experience with real system development and help them to appreciate the immediate applicability of what they have learned. On the other hand, to make clear the connection between requirements and tests [56], requirements were made explicit and detailed. Therefore, tests were also part of the submission, which included, along with the source code, a short report documenting the design rationale and a self-evaluation of the application. The assignment brief also outlined quality criteria that were relevant for the assessment of the work.

In an anonymous post-course evaluation survey (n=39), 95% of the students agreed (26%) or strongly agree (69%) that the assignment was relevant to the learning objectives. Moreover, the assignment was pre- and post-moderated by an external examiner, as part of the standard academic quality assurance process. Marks were also post-moderated.

In general, our quality criteria are inspired by the software quality models proposed by Boehm et al. [7] and McCall [45]. In our context, we assume that the quality of a student's work is proportional to the overall grade achieved, as a reflection of the ability to correctly fulfil assignment objectives, expressed in terms of software requirements. In practice, each set of requirements/features had a maximum score and each set was marked using the following criteria (borrowed from [3] for what concerns the programming style, and [22] which specifically defines a software quality model for mobile applications):

- completeness and effectiveness of the solution w.r.t. the requirements;
- correctness and accuracy w.r.t. the requirements;
- code structure and efficiency;
- UI: appropriate layout, consistency and usage of components;
- coverage and effectiveness of testing;

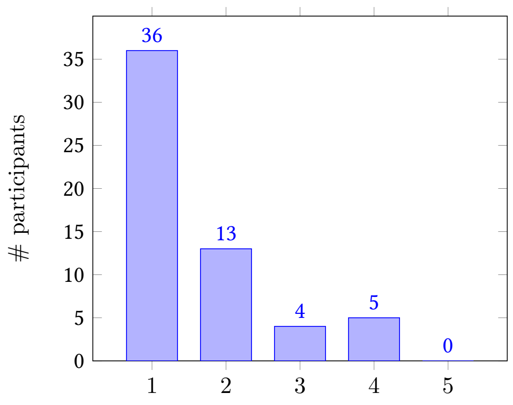

Fig. 3. Before taking this module, what was your knowledge of Android development? (1=very low,5=very high) – n=58, η=1.67, σ=0.93

- clarity and effectiveness of the documentation, including code comments.

4.2 Students' Feedback

In order to investigate how these scripts influenced the learning experience, a quantitative research approach [13] based on survey methodology was adopted. The online survey was designed using the LimeSurvey open-source software [40] and consisted of 7 questions including 6 quantitative and 1 qualitative (open-ended) questions. Participants were recruited sending individual email invitations to the 141 (81+60) undergraduate students enrolled in the "Android Mobile Development" module for two consecutive academic years (2016-17 and 2017-18). Each time, the survey was distributed five months after the completion of the module to reduce the chances of amplifying positive results. The questionnaire was designed to make possible linking the responses to the data extracted from the analysis of software artefacts previously submitted by the participants as part of the assignment task for this module. Answer were kept confidential, and data presented in this paper is published in an anonymised and aggregated form. All data has been collected and processed according with the research ethics policy of the institution. We received 58 (36+22) complete responses (response rate: 41.13%), while 11 (6+5) students just opened the questionnaire without filling it in.

4.2.1 Questionnaire. First of all, students were asked about their general knowledge of Android application development before the start of the module – a 5-point Likert scale, ranging from very low (1) to very high (5). Then a question specifically asked about the awareness of the possibility to build Android apps using the command-line tools. They were also asked if, during or after the module completion, they had ever used the scripts. If the answer was affirmative, they were offered the possibility to indicate for which activity/task they found the scripts useful choosing within a list of options (multiple answers allowed) with the option to further add activities/tasks proposed by the respondent. Next, the students were asked if they think they may

#	**Question**	**% yes**	**% no**
1	Before taking this module, did you know that Android apps could be built from the command line?	22.41	77.59
2	During or after this module, have you ever used the CMD scripts?	74.14	25.86
3	Do you think you might develop Android apps in the future?	81.03	18.97
4	In this case, do you think you might use the CMD scripts?	75.56	24.44

Table 1. Selected survey questions (*n*=58)

#	**Option** (multiple options allowed)	**% selected**
1	understand that Android apps can be built/cleaned/run from the command line	76.74
2	prepare the submission of my application for the assignment	72.09
3	build my project faster than with Android Studio	48.84
4	clean my project faster than with Android Studio	65.12
5	load faster my app on the emulator	44.19
6	check if my project was building or not	58.14
7	detect compiling errors more clearly than with Android Studio	27.91
8	check compliance of the environment configuration w.r.t the assignment requirements	32.59

Table 2. Question: "Could you please say if you found them useful to ... ?"

develop Android apps in the future, and if their answer was affirmative, they were asked if they would consider using the scripts. Finally, an open-ended question allowed the respondents to provide further comments and suggestions.

4.2.2 Results. The large majority of participants declared to have a very low (62.07%) or low (22.41%) prior knowledge of Android Development (Figure 3): only 22.41% of them knew that Android apps could be built from the command line (Table 1). We appreciate that it would have been better to formally collect answers to these questions at the beginning of the teaching term, but for practical reasons this was not possible at that point. However, it should be noted that these two questions were mainly asked to understand the demographics of the participants.

In terms of "*reaction*", 74.14% declared that they used the proposed scripts. The fact that not all participants used the scripts can be explained by a series of factors (inferred from the open-ended question): first of all, the scripts were presented as a (positive) alternative but the choice of using them was entirely left to the individual student. Additionally, there are users that in general do not feel comfortable with command-line tools, and they naturally prefer programs, like Android Studio, with a GUI neglecting CLI tools [59]. Finally, since the scripts were available only for Windows, the platform of the university labs, some students, using a different OS at home, were less interested in using them.

Overall (among those who used the scripts, multiple responses were allowed), the participants found the scripts useful (Table 2) to understand that Android apps can be built/cleaned/ran from the command line (76.74%), to prepare the submission of the application for the assignment (72.09%), to check if the project was building or not (58.14%), to clean (65.12%) and build (48.84%) the project faster than with Android Studio. The last figure may be apparently lower than expected compared with the "clean" option, because, in terms of speed, the difference in favour of CLI is remarkable (see Table 3). However, cleaning the project before submission was one of the requirements of the assignment (because it reduces considerably the size of the project), hence "clean" attracted more interest. Detecting compiling errors (27.91%) and checking the configuration of the environment (32.59%) attracted less interest.

More insights were offered by the last question (open-ended) where participants could detail their opinion. Overall, they gave a significant positive feedback. Some examples:

- "It was a great experience using the *cmd (CLI scripts)* because it give me the edge that I did not have, especially using *cmd* in order to compile, run or build an app because with *cmd* things were faster and easy than the Android Studio."
- "Easier, faster and more convenient way to run specific tasks that would otherwise be a bit inconvenient to run from A.Studio's convoluted menu."
- "They were good for building and cleaning the solutions and provided a quicker way of doing this."
- "Easy to use with just single line command. Less time consuming when cleaning and building a project."
- "They helped a lot with some of the more tedious tasks of needing to clean up regularly and saved time which was helpful."

However, some students confirmed their preference for GUI over CLI:

- "I think they are very useful however, I personally feel more comfortable using the UI to perform virtually any command it will allow me to perform."

Regarding the impact on the "*behaviour*", 81.03% of the survey participants think that in the future they might develop Android apps, and, among them, 75.56% think that they might use the scripts again. This is clearly an achievement as the initial knowledge of Android development was generally low.

Activity	***I**nteractive or **B**atch mode*		*η time (mm:ss)*			
	Android Studio	Scripts	Android Studio	Scripts	Diff	Gain %
open + build*	I	B	00:47	00:27	00:20	41.55
clean	I	B	00:11	00:08	00:03	21.12
(re)build	I	B	00:16	00:06	00:10	62.50
test	I	B	00:35	00:23	00:12	33.96
run	I	B	00:26	00:15	00:11	43.03
clean + zip	I	B	00:30	00:10	00:20	64.44
Total time			02:46	01:31	01:15	44.98
**applies only to Android Studio*						
Average Project Size (MB)			56.02	1.29	54.72	97.68

Table 3. Comparison of activities per single project: Android Studio vs scripts

In conclusion, most participants used the scripts and found them useful to develop Android applications but there are still some barriers to a complete adoption. As a reason for not using the scripts, participants essentially mentioned the unavailability for MacOS (6.90% of the overall respondents), and their preference for GUIs in any situation (1.72%). Some participants (3.45%) stated that they understand the benefits, but they did not use the scripts as they have their prior workflow already in place having developed for Android in the past.

4.3 Performance analysis

We carried out a performance evaluation regarding the usage of the scripts, from the students' point of view. In particular, we compared different activities (Table 3) completed with Android Studio and with the scripts. The machine used was a Windows 10 laptop, with 16GB RAM and Intel Core i7 4700HQ @ 2.40GHz processor, Android Studio 2.3.3, Android SDK API 23, Gradle 4.1, AVD Nexus 5 API23. Overall, we observed that the scripts run faster with significant gains obtained in all the tasks. It should be noted that a *clean+zip* function does not exist in Android Studio, so we considered cleaning a project and zipping by hand versus running the `zip_single` script. Interestingly, after closure, a non-trivial project can use around 40-50 times more space in Android Studio than a project zipped with the scripts. This has a significant impact when students need to submit their files or transfer their projects from one machine to another. In terms of resources, the scripts tend to perform similarly even if the RAM is halved from 16GB to 8GB, while Android Studio performance noticeably degrades as the amount of volatile memory is reduced, as it was the case of the machines used in the university labs.

4.4 Assessment Data Analysis

Another objective of the evaluation is to investigate if the scripts had a "*learning*" impact on the student performance. In other words, has the use of the scripts allowed the students to produce a qualitatively better work, according to the criteria defined in §4.1?

In order to answer this question, our approach is to partition the students in two groups (those who have used the scripts or those who have not) and see how they compare in terms of module results. Would their grades be similar or different?

First of all, it should be noted that this is a quasi-experiment rather than a true experiment, as we were working in a real education environment and the students using the scripts were a self-selecting group. Our analysis involves data from the assignment, in particular the grading of the applications submitted by all students, including those who have not participated to the questionnaire. This has the advantage of not only having more data for our analysis, but also of being more precise, as we are able to tell who used the scripts for the preparation of the assignment (artefact), rather than relying on participants declaring having used the scripts at one point in the past (questionnaire).

To distinguish the two groups, we considered the following criterion: if the source code contains at least one of the two folders (`/build` and `/app/build`) or the file `local.properties` the student has not used the scripts. The two `build` folders are created when a project is built, and removed when the project is cleaned with the script `clean_single` or prepared for submission with `zip_single`. However, the students who have only used Android Studio but not the scripts to prepare their submission, will have these folders included in the submitted compressed file (unless manually deleted) because of the automatic build feature of the IDE. Moreover, the script `zip_single` also removes the `local.properties` file. This is because the file stores the location of the local Android SDK. When moving a project to another machine, the file needs to be updated and, in Android Studio, this prompts a warning message and recreates the file silently. `local.properties` is not removed by the Android Studio clean process, therefore, its absence is extremely unlikely if the scripts have not been used for the preparation of the submission.

Table 4 shows the number of valid submissions (136 out of 141 enrolled) during the first session for each cohort (2017 and 2018) and the aggregate data for different groups (mark range: 0–100, pass threshold = 40). The demographics of the students was the following:

- *Sex*: male 91.2% – female 8.8%
- *Age*: 20-24 90.4% – 25-29 8.1% – 30 and over 1.5%
- *Domicile*: UK 76.5% – EU 12.5% – International 11.0%
- *Degree Course*: Computer Science 49.3% – Computing 28.7% – Games Software Development 10.3% – Computer Systems Engineering 8.1% – Other or not known: 3.6%

We denote with n the number of students, η the mean grade, σ the standard deviation and Δ the difference between the means. It should be noted that all survey

Group	*Cohort*	*n*	η	σ
All Submissions	1+2	136	63.36	22.21
Passed		120	68.51	18.10
Survey participants		58	71.52	21.11
All Submissions	1	78	65.01	20.37
Passed		71	68.79	17.09
Survey participants		36	73.75	19.27
All Submissions	2	58	61.14	24.48
Passed		49	68.01	19.64
Survey participants		22	67.86	23.83

Table 4. Grades of students enrolled in the course (2 cohorts) (max grade=100)

Used scripts	*Cohort*	*n*	η	σ	*Wilcoxon Mann-Whitney test*
Yes	1+2	72	72.39	18.57	W=3437.5 p=7.71e-07
No		64	53.20	21.69	
		Δ	19.19		
Yes	1	43	73.70	18.21	W=1166.5 p=3.214e-05
No		35	54.34	17.81	
		Δ	19.36		
Yes	2	29	70.45	19.26	W=608 p=0.00361
No		29	51.83	25.87	
		Δ	18.62		

Table 5. Grades data analysis for different groups of participants/students

participants have submitted in the first session and their average mark (η) is higher than the overall group.

Since the distribution of these groups is not normal (as all the other ones considered here) we used the non-parametric two-tailed Wilcoxon/Mann-Whitney test to accept or reject the null hypothesis H_0: the two groups do not differ with respect to the considered criterion (Table 5). In this case, H_0 is rejected (p-value < 0.05) and therefore there is a statistical difference between the two groups. The one that according to our criterion has used the scripts has achieved on average (Δ=19.19) more marks than the other one. The same analysis repeated on the two cohorts individually gives similar results.

However, at this stage, since the groups considered are self-selected, we cannot exclude that the more technically savvy students might have been more prone to take advantage of the scripts, or, in general, other factors like motivation might have influenced the choice of using the scripts or not.

Therefore, considering other data available, we computed the two-tailed Fisher Exact Probability test (confidence interval 95%), a statistical significance test used in the analysis of contingency tables, for the following null hypothesis: relative proportions of one variable (having used the scripts or not) are independent of the second variable. We considered "*prior knowledge of Android development*" (p-value= 0.2207) and "*prior knowledge that Android apps could be built from the command line*" (p-value= 0.1912). We also considered an issue like motivation, e.g. students who think they "*might develop Android apps in the future*" (p-value= 0.7294). Interestingly, we also considered the overall performance of the students in the degree course according to the British degree classification – (1st) 70 to 100, (2:1) 60 to 69, (2:2) 50 to 59, (3rd) 40 to 49 – (p-value= 0.4899). In all cases, we could not reject the null hypothesis (since p-value > 0.05). Therefore, we can conclude that it is highly unlikely that the factors considered (in particular that high-performing students may have been more inclined to use the scripts than the entire cohort) may be statistically correlated with the choice of using the scripts.

4.5 Interpretation of results

The results presented can be interpreted in light with the existing literature. A first question is whether the command-line tools can be beneficial for learning programming. Several empirical and quantitative analysis studies [12, 17, 18] answer positively to this question. Chen and Marx [12] have reported the benefits of using the command-line interface tools in Java programming courses before introducing GUI-based IDEs and they decided to reverse, in the first weeks of their introductory programming course, from Eclipse to command-line tools.

According to Dillon et al. [16] this enables learners to develop better mental models for programming because of the limited range of operations and the fact that users cannot skip steps in the workflow. In fact, CLI tools for programmers allow not only to the learn the language syntax and problem solving skills, but also (and this relevant in our case) help students to get a clear understanding of programming procedures like compilation, execution, and editing [12, 17].

Chen and Marx [12] report that this also increases the level of confidence in the usage of the IDEs when they are introduced. It should be noted that the transition from command line to IDE is rather easy, but the other way around could be more challenging [18]. The danger is that students initially exposed only to IDEs may get the impression that clicking a button makes their applications magically work. For example, we observed that students using exclusively Android Studio could get confused, failing to understand precisely the point of failure in a sequence of actions (e.g. clean/build/run) interpreting the sequence as an atomic (one click) action; in the worst case they were even unsure if an error has occurred.

Kuusinen [37] considered, in the context of software development, flow experience, a the state of concentration in which the individual is fully absorbed by an activity.

We think our scripts can contribute to the satisfaction of the three conditions essential to achieve a flow state [14]:

- involvement in an activity with a clear set of goals and progress: each script performs a clear set of tasks in a well defined workflow;
- tasks must have clear and immediate feedback: e.g. the scripts output clearly any anomaly condition;
- confidence in one's ability to complete a task based on a good balance between challenges and perceived skills: our questionnaire data show that users found the scripts useful to complete the tasks, adequate to their skill level.

Another crucial aspect is that in the Android environment the code is executed not on the developer machine but on a virtual or external device. This can create some confusion in novice Android developers. Therefore, as highlighted by Dillon et al. [18], even though IDEs can lower the learning curve for operations, they can potentially limit or mislead the mental representation of programming procedures.

Some of cited studies [12, 17] consider mostly novice programmers but, in our case, students were relatively more experienced as they took this module in their third and final year of an undergraduate computing degree. They are rather newcomer programmers [55] as they are new to the programming environment and the complexity of Android Studio can be overwhelming.

As discussed, enabling students to work with professional tools is one of our learning objectives and Android Studio being the industry de-facto standard for Android development, makes the usage of this IDE almost unavoidable.

Therefore, since our scripts are not a replacement for the IDE, but rather complementary tools to carry out programming procedures more simply and more efficiently, we found appropriate to introduce the scripts along with the IDE. As these scripts require students to follow well-defined workflows, they help to better understand and more precisely control the most crucial programming procedures performed during the development cycle. Moreover, it should be considered that such procedures can also be more time consuming if done with the IDE given the complexity of the environment and hardware limitations (see §3.1 and §5). For examples, we noted that the Gradle build process under Android Studio can become slow and unresponsive, in particular when the RAM available is 8GB or less. In contrast, this is much less likely to happen if the build process is performed from the CLI, where it is also clearer to understand whether the process is progressing or not.

5 EVALUATION OF THE ADMIN'S SCRIPTS

We discuss now how the scripts can be useful from the tutor's point of view to streamline the marking process and keep the teaching material up-to-date, adapting to the frequent changes of the Android platform.

5.1 Marking Procedure

As discussed in §2, the submission process and the assessment of mobile apps can be challenging. In our case, a student's submission consisted of an Android project

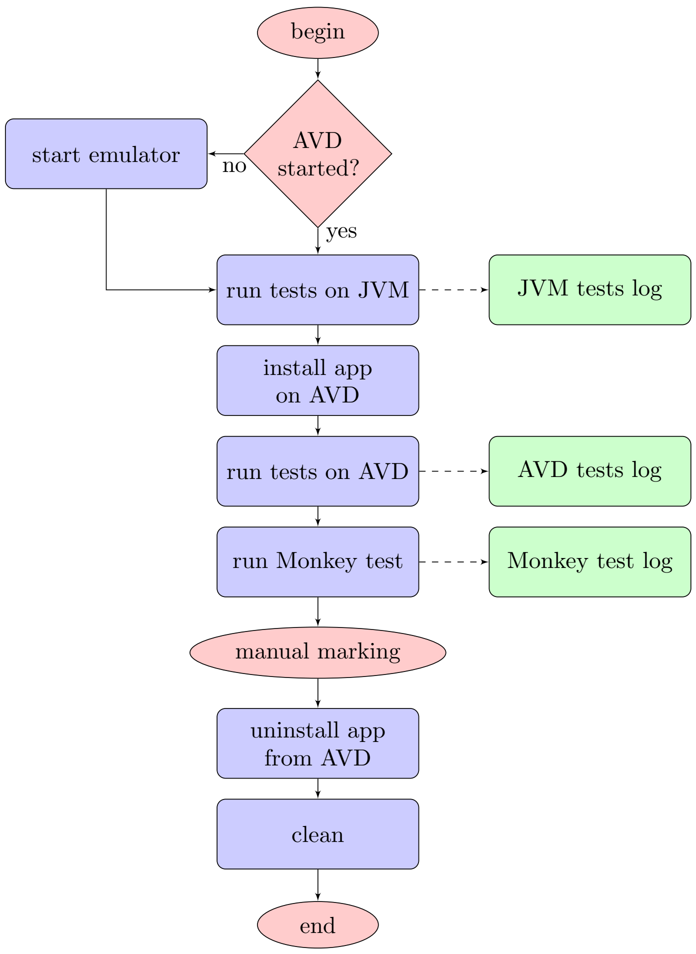

Fig. 4. Marking workflow (script `mark_single`)

(source code files and resources) and a report submitted online inside a compressed file. The typical work of a tutor/marker begins (Figure 2) by downloading the students' submissions files from the online repository to a local folder.

A possible issue is that some students do not properly follow the instructions for submission (e.g. configuration of the "target" platform). Therefore, the marker has to make some sanity checks and perform refactoring operations which, if done manually on a large group like the one we considered here ($n = 136$), may require hours of tedious and repetitive work. The scripts `refactor` allows the fixing of issues regarding the project folder structure, while the script `migrate` helps to align the project to the "target" configuration.

Moreover, before running, all projects need to be cleaned and built, as meta-data and temporary files from other systems can interfere with the correct compilation of the application. With Android Studio, given the interactive nature of the GUI, this

Activity	***I**nteractive or **B**atch mode*		*η time (mm:ss)*			
	Android Studio	Scripts	Android Studio	Scripts	Diff	Gain %
refactor	I	B	00:25	00:10	00:15	60.00
migrate	I	B	01:41	00:07	01:34	93.07
clean+build	I	B	00:29	00:23	00:06	20.69
test	I	I**	00:35	00:23	00:12	33.96
*mark**	I	I	01:26	00:14	01:12	83.72
Interactive time			04:36	00:37	03:55	87.17
Batch time			00:00	00:40	–	–
Total time			04:36	01:17	03:19	72.10

* excluding test ** can also run in batch mode

Table 6. Comparison of activities per single project: Android Studio vs scripts (Marking)

will also take further time as the marker needs to import each project and wait for the completion of the operation.

Instead, no marker's interaction is needed if this is done in batch mode (scripts `clean` and `build`). It should be noted, that if the results of the batch processing are logged, the marker is able, at any time, to see the progress of the operations. It also possible to check, for any submission, if the building process fails. This kind of information can be used as a component of the marking as it indicates properties of the submission. For example, the build process also run a Lint-like tool, which performs static checks on the source code and can provide hints for potential structural problems in the code that could impact on the reliability and quality of the application. This also relates with the software quality criteria discussed for the assignment design (§4.1).

Afterwards, the marker needs to run each project individually, and grade it according to the marking criteria. A further time efficiency gain can be achieved, using a procedure (script `mark_single`, Figure 4) allowing to automatically install and run the app in the AVD, run the tests on the JVM and AVD, display the output of the tests, open with an editor the relevant files of the source code that the marker needs to inspect.

Automatising this process, can save further time, avoiding a large number of tedious repetitive manual interactions with the GUI of the IDE, and the scripts were indeed useful to shift precious time during the marking process from mundane operations to the actual marking.

5.2 Performance Analysis

In order to test this claim, we ran an experiment on a Windows 10 laptop, with 16GB RAM and Intel Core i7 4700HQ @ 2.40GHz processor, Android Studio 2.3.3, Android SDK API 23, Gradle 4.1, AVD Nexus 5 API23. We compared the marking process for a sample of projects (Table 6), and found that, the automatic procedure allows to save 72% of the time. In particular, the interactive time can be reduced by 87%. In practice, the marker can just run the batch part, collect the output at any time and then start the interactive part of the marking. In this case, using the script is still advantageous as 83% of the time spent in trivial operations (opening, running, closing, etc.) can be saved.

On average, the time saved for each project is more than 3 minutes. Although the exact figure may vary according with the different versions of the tools (for example, we noticed that the *clean+build* time was much higher in Android Studio 2.1.3/Gradle 2.14.1 then in Android Studio 2.3.3/Gradle 4.1), we believe that there is a strong indication that the scripts can save a considerable amount of time to the marker (e.g., more than 7.5 hours, for 136 submissions) along with a lot of repetitive work. In general, the scripts help to standardise the marking process, avoiding errors and shortcomings typical of interactive procedures. The external marker who used this procedure for the TNE instances of this module at Botswana Accountancy College in Gaborone and Francistown, agreed on the beneficial impact in terms of efficiency and quality of the process.

5.3 Keeping Tutorial Material Up To Date

As mentioned, the Android landscape is very dynamic with frequent changes to the API. Approximately, every year a major new version of the OS is released along with two or three new levels of the API. The speed of change has been identified [9, 53, 56] as a crucial challenge for instructors because it requires revising the teaching material to keep the course up to date.

In particular, due to the complexity of the technology, demonstration apps and exercises not only require a consistent amount of time to be prepared but also need to be periodically updated to adapt the changes of the platform: new versions of the API, build tools, Android Virtual Devices, IDE, etc. On itself, the migration is in general complex and error-prone [38], and the update process can also be tedious and time consuming because the teaching material for an entire course can comprise many dozens of apps and each app needs to be validated (built and run) against the new "target" configuration, to ensure that no issues arise during lectures or tutorials.

To that purpose, the admin's scripts can be very useful to automatically migrate and validate an entire library of applications from one version of the API/SDK to another one (`migrate` script). This can be done with a procedure similar to the one depicted in Figure 2. In order to support a new version, the tutor needs to create a new configuration file, specifying parameters like versions of the SDK (min|compile|target), build tools, Gradle and major libraries like the `com.android.support` and `com.google.android.gms:play-services`.

Although the procedure in not bulletproof, as there is no guarantee that purely syntactic changes in Android projects (e.g. manifest file and source code) are sufficient for the application to compile and run successfully, in practice, our experience with this process has been positive as we have never faced any critical issue.

In fact, usually, obsolete API classes and methods are not suddenly removed from the Android libraries but are rather labelled as "deprecated" for a period of time which usually comprised the release of at least a few new revisions of the API before being phased-out (the so-called *deprecate-replace-remove* cycle [39]). This allows time for developers to smoothly migrate to a new version and, in the interim, tutors can still use the previous version of an application (migrated to the new configuration) in their classes.

6 CONCLUSION

In this work, we presented a script-based approach for native Android application development that complements existing tools (e.g. Android Studio, Android SDK). The evaluation considered the students' feedback, the data analysis of the module outcome, software artefacts and the application of the scripts during the marking process. In all these areas, the evaluation suggests that the scripts had a positive impact both from the student and instructor's point of view.

As these scripts complement Android Studio, students are still exposed to real-world tools used in the industry, and some former students working as professional app developers reported they were still using the scripts. Another positive aspect is that these scripts facilitate the delivery of courses in situations where access to powerful hardware resources can be limited, like in developing countries.

Although the usage of CLI scripts for software development is not new, such approach was not previously considered for Android. To the best of our knowledge, this is the first attempt to integrate CLI scripts in an Android development pedagogy. The acceptance of CLI scripts among students was mostly positive, and evidence suggests the using the scripts may have contributed to improve the student performance in this specific module, regardless of their overall performance in the full degree course.

Recommendations. Based on our experience, the choice of the Android development tools is a critical aspect for the success of a teaching strategy. One challenge is how to help students to develop a clear conceptual workflow. This is addressed by the scripts allowing for a simple and systematic way to handle the complexity of Android project configuration and management. Another issue is how to provide a practical mechanism to distribute and submit Android projects. The size of a simple project could be 50-60 MB, but with the `zip_single` script (cleaning and zipping a project, an operation which does not exist in Android Studio) it is possible to shrink the size to just 1 MB saving space and reducing time for upload/download. Moreover, the same script solves issues related with moving a project from one machine to another (e.g. hard-coded absolute paths).

Tutors should carefully consider the amount of freedom to grant students in using different versions of tools/API as this can impact negatively in different phases of the development and marking process. In fact, incompatibilities may break the application. For that reason, in our approach, we identified each year a reference configuration (indicatively, a recent stable version of Android Studio and an API level that allows to cover at least 95% of currently active devices), and provided students not only with the instructions to set the environment up but also a script (`chkSDK`) to check whether their configuration complies with the reference one. Our experience suggests that having a standard configuration not only proved to be of utmost importance in dealing with large cohorts of students, but also encouraged students to develop a greater control of the development environment, an important professional skill for every developer. Additionally, the script for checking the compliance of the configuration was also very useful to set up labs where courses were franchised.

Future research directions. Further research should consider several directions. An important one would be to investigate if other complex development environments could benefit from a script-based approach. An obvious candidate would be the native mobile application development for iOS, but it could also be worth studying other types of app development paradigms [48]. Additional research may also consider analysing tool log/usage files (e.g. command-line and IDE) in order to understand more precisely the behaviour of developers, their progress, difficulties and the construction of mental models [23]. Moreover, such studies should also include professional developers rather than just students, because of their different degree of expertise and motivation.

In this paper, we also presented our work towards the automation of the marking process. Although relevant information can be gathered from log files (as we indeed filtered them with grep-like commands), most of the "meaningful" marking was still done manually, running the application, inspecting the logs and checking if the requirements were fulfilled, according to the marking criteria.

From the tutor's perspective, the scripts also allow for a more efficient management of the teaching material and configuration migration. Therefore, the scripts can offer tutors a practical solution in their delivery of Android development courses.

A significant step would be moving further towards the automatic grading of Android assignments. There is a considerable number of tools that have been developed for the automated assignment grading and feedback, reviewed and classified by various review papers [10, 31, 34]. In particular, Wilcox [60] identified a set of testing strategies for automated grading.

A limited number of tools have been proposed for the automated grading of Android assignments. RoboLIFT [4] is a library that is meant to facilitate the testing of Android applications to a level adequate to students. It integrates with Web-CAT [19] a platform for automated grading. The approach of RoboLIFT is based on Test-Driven Development (TDD) that requires to write tests in advance and then implements the code that make them pass. This is different from the conventional approach where code is written first and then tested. Unfortunately, RoboLIFT has some limitations, notably it works with applications with just a single Android activity. Therefore, this does not make it suitable for most apps which are usually composed of multiple activities. Madeja and Porubän [42] considered the most common student mistakes and designed a system to detect them through testing. They defined milestones and identified a set of tests expected to pass at that stage. However, in order to allow tests to run successfully on student code, they had to impose restrictions to the UI (including on identifiers names), and when they did not put any further restriction in the last phase of development, not to limit the creativity of student, this greatly reduced the success rate. Bruzual et al. [8] presented a system for automated assessment of Android exercises carried out by running exercise-specific unit tests on the APK file, relieving the tutor from the need to compile the student's submission. However, while this approach seems to scale well, ignoring the source code can limit the possibility to offer more insightful feedback to students.

Although any automated grading system for Android is likely to quickly become obsolete due to the frequent changes to the Android development environment, in our case, a first step could be to further automatise the analysis of the logs produced during the marking procedure (§5) extracting and processing automatically information relevant for the assessment, and provide a core set of test cases along with the assignment specification. Moreover, it should be possible to run tools that compute quality parameters on the source code that can be included in the assessment score.

ACKNOWLEDGMENTS

Part of this work was done while the author was working at the University of Sunderland. The author wishes to thank Sophie Cormack, James Fairbairn, Sharon McDonald and the anonymous reviewers for their constructive feedback.

REFERENCES

[1] Seiko Akayama, Birgit Demuth, Timothy C Lethbridge, Marion Scholz, Perdita Stevens, and Dave R Stikkolorum. 2013. Tool Use in Software Modelling Education.. In *EduSymp@ MoDELS*.

[2] Abrar Al-Heeti. 2019. Android is on over 2.5 billion active devices. https://www.cnet.com/news/android-is-on-over-2-5-billion-active-devices//. Online; accessed 04 December 2019.

[3] Kirsti Ala-Mutka, Toni Uimonen, and Hannu-Matti Jarvinen. 2004. Supporting students in C++ programming courses with automatic program style assessment. *Journal of Information Technology Education: Research* 3 (2004), 245–262.

[4] Anthony Allevato and Stephen H. Edwards. 2012. RoboLIFT: Engaging CS2 Students with Testable, Automatically Evaluated Android Applications. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. ACM, New York, NY, USA, 547–552. https://doi.org/10.1145/2157136.2157293

[5] Android Studio. 2020. Android Studio Release Notes. https://developer.android.com/studio/releases/. Online; accessed 19 June 2019.

[6] Mordechai Ben-Ari. 2001. Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching* 20, 1 (2001), 45–74.

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

[8] Daniel Bruzual, Maria L. Montoya Freire, and Mario Di Francesco. 2020. Automated Assessment of Android Exercises with Cloud-native Technologies. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2020, Trondheim, Norway, June 15-19, 2020*, Michail N. Giannakos, Guttorm Sindre, Andrew Luxton-Reilly, and Monica Divitini (Eds.). ACM, 40–46. https://doi.org/10.1145/3341525.3387430

[9] Barry Burd, João Paulo Barros, Chris Johnson, Stan Kurkovsky, Arnold Rosenbloom, and Nikolai Tillman. 2012. Educating for mobile computing: addressing the new challenges. In *Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups*. ACM, 51–63.

[10] Julio C. Caiza and José María del Álamo Ramiro. 2013. Programming assignments automatic grading: review of tools and implementations. In *7th International Technology, Education and Development Conference (INTED2013)*. 5691–5700. http://oa.upm.es/25765/

[11] Alexander Chatzigeorgiou, Tryfon L Theodorou, George E Violettas, and Stelios Xinogalos. 2016. Blending an Android development course with software engineering concepts. *Education and Information Technologies* 21, 6 (2016), 1847–1875.

[12] Zhixiong Chen and Delia Marx. 2005. Experiences with Eclipse IDE in Programming Courses. *J. Comput. Sci. Coll.* 21, 2 (Dec. 2005), 104–112. http://dl.acm.org/citation.cfm?id=1089053.1089068

[13] John W Creswell. 2013. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications.

[14] M. Csikszentmihalyi, S. Abuhamdeh, and J. Nakamura. 2005. *Handbook of competence and motivation*. New York: Guilford Press, Chapter "Flow", 598–698.

[15] Edsger W Dijkstra. 1989. On the cruelty of really teaching computing science. *Commun. ACM* 32, 12 (1989), 1398–1404.

[16] Edward Dillon, Monica Anderson, and Marcus Brown. 2012. Comparing mental models of novice programmers when using visual and command line environments. In *Proceedings of the 50th Annual Southeast Regional Conference*. ACM, 142–147.

[17] Edward Dillon, Monica Anderson-Herzog, and Marcus Brown. 2012. Studying the novice's perception of visual vs. Command line programming tools in CS1. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 56. SAGE Publications Sage CA: Los Angeles, CA, 605–609.

[18] Edward Dillon, Monica Anderson-Herzog, and Marcus Brown. 2014. Teaching students to program using visual environments: Impetus for a faulty mental model? *Journal of Computational Science Education* 5, 1 (2014), 1–2.

[19] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. *SIGCSE Bull.* 40, 3 (June 2008), 328–328. https://doi.org/10.1145/1597849.1384371

[20] César Fernández, María Asunción Vicente, M Mar Galotto, Miguel Martinez-Rach, and Alejandro Pomares. 2017. Improving student engagement on programming using app development with Android devices. *Computer Applications in Engineering Education* 25, 5 (2017), 659–668.

[21] Rita Francese, Carmine Gravino, Michele Risi, Giuseppe Scanniello, and Genoveffa Tortora. 2015. Using Project-Based-Learning in a mobile application development course - An experience report. *Journal of Visual Languages & Computing* 31, Part B (2015), 196–205. https://doi.org/10.1016/j.jvlc.2015.10.019

[22] Dominik Franke, Stefan Kowalewski, and Carsten Weise. 2012. A mobile software quality model. In *Quality Software (QSIC), 2012 12th International Conference on*. IEEE, 154–157.

[23] Markus Fuchs, Markus Heckner, Felix Raab, and Christian Wolff. 2014. Monitoring students' mobile app coding behavior data analysis based on IDE and browser interaction logs. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*. IEEE, 892–899.

[24] Mercedes Gómez-Albarrán. 2005. The Teaching and Learning of Programming: A Survey of Supporting Software Tools. *Comput. J.* 48, 2 (2005), 130. https://doi.org/10.1093/comjnl/bxh080

[25] Mark H. Goadrich and Michael P. Rogers. 2011. Smart Smartphone Development: IOS Versus Android. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. ACM, New York, NY, USA, 607–612. https://doi.org/10.1145/1953163.1953330

[26] Gradle. 2020. Gradle Build Tool. https://gradle.org. Online; accessed 19 June 2020.
[27] Tor-Morten Gronli, Jarle Hansen, Gheorghita Ghinea, and Muhammad Younas. 2014. Mobile application platform heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on.* IEEE, 635–641.
[28] Michael Halper. 2014. Using Android As a Platform for Programming in the IT Curriculum. In *Proceedings of the 15th Annual Conference on Information Technology Education* (Atlanta, Georgia, USA) *(SIGITE '14).* ACM, New York, NY, USA, 127–132. https://doi.org/10.1145/2656450.2656461
[29] Dongjie He, Lian Li, Lei Wang, Hengjie Zheng, Guangwei Li, and Jingling Xue. 2018. Understanding and detecting evolution-induced compatibility issues in Android apps. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering.* ACM, 167–177.
[30] IDC. 2020. Smartphone Market Share – Updated: 02 Apr 2020. https://www.idc.com/promo/smartphone-market-share/os. Online; accessed 19 June 2020.
[31] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '10).* ACM, New York, NY, USA, 86–93. https://doi.org/10.1145/1930464.1930480
[32] Ivaylo Ilinkin. 2014. Opportunities for Android Projects in a CS1 Course. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14).* ACM, New York, NY, USA, 615–620. https://doi.org/10.1145/2538862.2538983
[33] Tony Jenkins. 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, Vol. 4. 53–58.
[34] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2016. Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16).* ACM, New York, NY, USA, 41–46. https://doi.org/10.1145/2899415.2899422
[35] Donald L Kirkpatrick and JD Kirkpatrick. 2006. *The four levels: an overview.* Berret-Koehler Publishers San Francisco, Calif, USA. 26–35 pages.
[36] Stefan Koch and Markus Kerschbaum. 2014. Joining a smartphone ecosystem: Application developers' motivations and decision criteria. *Information and Software Technology* 56, 11 (2014), 1423–1435.
[37] Kati Kuusinen. 2016. Are software developers just users of development tools? Assessing developer experience of a graphical user interface designer. In *Human-Centered and Error-Resilient Systems Development.* Springer, 215–233.
[38] Maxime Lamothe and Weiyi Shang. 2018. Exploring the use of automated API migrating techniques in practice: an experience report on Android. In *Proceedings of the 15th International Conference on Mining Software Repositories.* 503–514.
[39] Li Li, Jun Gao, Tegawendé F Bissyandé, Lei Ma, Xin Xia, and Jacques Klein. 2018. Characterising deprecated android apis. In *Proceedings of the 15th International Conference on Mining Software Repositories.* ACM, 254–264.
[40] LimeSurvey Team. 2020. LimeSurvey: an open source survey tool. http://www.limesurvey.org. Online; accessed 19 June 2020.
[41] Andrew Luxton-Reilly, Ibrahim Albluwi, Brett A Becker, Michail Giannakos, Amruth N Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, Claudia Szabo, et al. 2018. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education.* ACM, 55–106.
[42] Matej Madeja and Jaroslav Porubän. 2018. Automated testing environment and assessment of assignments for Android MOOC. *Open Computer Science* 8, 1 (2018), 80–92.

[43] Ference Marton and Roger Säljö. 1976. On qualitative differences in learning: Outcome and process. *British journal of educational psychology* 46, 1 (1976), 4–11.

[44] Victor Matos and Rebecca Grasser. 2010. Building applications for the Android OS mobile platform: a primer and course materials. *Journal of Computing Sciences in Colleges* 26, 1 (2010), 23–29.

[45] J McCall. 1977. Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager, volume 1-3. *General Electric, November* 130 (1977).

[46] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students. *SIGCSE Bull.* 33, 4 (Dec. 2001), 125–180. https://doi.org/10.1145/572139.572181

[47] T. McDonnell, B. Ray, and M. Kim. 2013. An Empirical Study of API Stability and Adoption in the Android Ecosystem. In *Proc. IEEE Int. Conf. Software Maintenance*. 70–79. https://doi.org/10.1109/ICSM.2013.18

[48] Robin Nunkesser. 2018. Beyond web/native/hybrid: a new taxonomy for mobile app development. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, MOBILESoft@ICSE 2018, Gothenburg, Sweden, May 27 - 28, 2018*, Christine Julien, Grace A. Lewis, and Itai Segall (Eds.). ACM, 214–218. https://doi.org/10.1145/3197231.3197260

[49] Luca Pascarella, Franz-Xaver Geiger, Fabio Palomba, Dario Di Nucci, Ivano Malavolta, and Alberto Bacchelli. 2018. Self-reported activities of Android developers. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, MOBILESoft@ICSE 2018, Gothenburg, Sweden, May 27 - 28, 2018*, Christine Julien, Grace A. Lewis, and Itai Segall (Eds.). ACM, 144–155. https://doi.org/10.1145/3197231.3197251

[50] Evan W Patton, Michael Tissenbaum, and Farzeen Harunani. 2019. MIT App Inventor: Objectives, Design, and Development. In *Computational Thinking Education*. Springer, 31–49.

[51] Roy D Pea and D Midian Kurland. 1984. On the cognitive effects of learning computer programming. *New ideas in psychology* 2, 2 (1984), 137–168.

[52] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A Survey of Literature on the Teaching of Introductory Programming. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education* (Dundee, Scotland) *(ITiCSE-WGR '07)*. ACM, New York, NY, USA, 204–223. https://doi.org/10.1145/1345443.1345441

[53] Derek Riley. 2012. Using mobile phone programming to teach Java and advanced programming to computer scientists. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 541–546.

[54] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: A review and discussion. *Computer science education* 13, 2 (2003), 137–172.

[55] Susan Elliott Sim and Richard C Holt. 1998. The ramp-up problem in software projects: A case study of how software immigrants naturalize. In *Proceedings of the 20th international conference on Software engineering*. IEEE, 361–370.

[56] Jonathan Sprinkle. 2011. Teaching Students to Learn to Learn Mobile Phone Programming. In *Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11* (Portland, Oregon, USA) *(SPLASH '11 Workshops)*. ACM, New York, NY, USA, 261–266. https://doi.org/10.1145/2095050.2095094

[57] Neena Thota and Richard Whitfield. 2010. Holistic approach to learning and teaching introductory object-oriented programming. *Computer Science Education* 20, 2 (2010), 103–127.

[58] Phillip Treweek. 1996. Comparing interfaces: should we assume that ease of use influences users' preference?. In *Computer-Human Interaction, 1996. Proceedings., Sixth Australian Conference on*. IEEE, 159–160.

[59] Antony Unwin and Heike Hofmann. 1999. GUI and Command-line - Conflict or Synergy? In *Proceedings of the 31st Symposium on the Interface: models, predictions, and computing,*

Schaumburg, Illinois, June 9 - 12, 1999, Kenneth Berk and Mohsen Pourahmadi (Eds.). Interface Foundation of North America, 246–253.

[60] Chris Wilcox. 2016. Testing strategies for the automated grading of student programs. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM, 437–442.

[61] Xiaohong Yuan, K. Williams, S. McCrickard, C. Hardnett, L. H. Lineberry, K. Bryant, Jinsheng Xu, A. Esterline, Anyi Liu, S. Mohanarajah, and R. Rutledge. 2016. Teaching mobile computing and mobile security. In *Proc. IEEE Frontiers in Education Conf. (FIE)*. 1–6. https://doi.org/10.1109/FIE.2016.7757365