Proceedings of the Federated Conference on Computer Science and Information Systems pp. 903–912 DOI: 10.15439/2016F432 ACSIS, Vol. 8. ISSN 2300-5963

A blended learning model for practical sessions

Nuno Barreiro, Carlos Matos
Department of Computer Science,
Royal Holloway, University of London,
Egham Hill,
TW20 0EX, UK
Email: {Nuno.Barreiro, Carlos.Matos}@rhul.ac.uk

***Abstract*—We describe an education model that was developed and put in place to improve student engagement and attainment in a first year undergraduate programming course.**

The work is founded in a checkpoint-based formative assessment experiment undertaken for two years, the success of which is analysed in this document. The results provide evidence leading to a move towards a blended model of education, which requires the design of a software application to support the system. We present the main features of that application, covering aspects that range from traditional approaches and established delivery methods, to e-learning and MOOCs with, for instance, gamification.

This blended model of education fosters the development of a teaching practice that adapts to student diversity through informed teaching.

I. Introduction

FIRST year higher education undergraduate teaching is normally challenging for most areas: students face issues when transitioning from the school system; the class as a whole presents an inhomogeneous skill diversity. This is certainly the case in computer science, an area that attracts candidates with many different backgrounds. A particularly challenging topic within the discipline is the induction of students to programming [1], [2]: students taking a first year undergraduate course in that subject range from those who have never seen a line of code, to those who have completed software projects at school.

This document describes and evaluates a formative assessment experiment undertaken for two years in such a first-year course: the practical sessions were radically transformed by the introduction of a *checkpoint system*. The main goal of this new approach was to improve student engagement and attainment, but other objectives were also taken into consideration when designing the system.

Lab sheets with strategically placed low granularity checkpoints drive the practical sessions, which are supported by teaching assistants (TAs). Their role is to validate the checkpoints, provide help and feedback to the students, and gather data. Updated on a weekly basis, a checkpoints map measures how students progress through the course, and is used to inform teaching: early failure triggers quick remedial actions; high achievers are provided with challenging material; the contents of the sessions are adapted weekly to the overall pace and performance of the class. The experience of running this model for two years proved successful, with high student engagement and very positive feedback both from academic staff and students.

The analysis of the experiment has led to the development of a blended learning model to support practical sessions. Blended learning [3], [4], [5] uses both paradigms of traditional brick-and-mortar teaching, and online digital technologies, taking stock of techniques proven successful in e-learning and MOOCs, for example in the context of primary school mathematics education [6].

The course already uses Moodle to publish all its material, namely the weekly lab sheets used in practical sessions. The approach we propose includes extensions based on gamification [7] and a semi-automation of the checkpoint validating process. This is achieved through a software application, which design is described in this document.

The remainder of this paper is organised as follows: Section II provides context to both model and experiment, which are described in Section III. Section IV details the qualitative analysis of the experiment, and leads into a discussion on moving towards a blended model of education, presented in Section V. The checkpoint system is supported by an application described in Section VI. Section VII discusses related work, and Section VIII presents the conclusions.

II. Context

A. *Teaching first year students*

Formative assessment plays a major role in teaching computer science at our department, in particular when it comes to programming. Students start the degree with a great diversity of individual skills, which adds to the challenge of getting all of them to the end of first year with a similar body of knowledge. Treating that range of ability as an homogeneous body raises lack of engagement at both ends of the spectrum, which has been observed during many years of teaching experience in the department. As such, we need tools that help us both address failure at a very early stage, and encourage students who have a high level of familiarity with the subject to move on to more challenging material.

Each course delivery takes that diversity into account, to some extent. However, when it comes to first-year practical sessions, this aspect plays a major role: those sessions are, for many students, the first contact with written computer code. Students with different backgrounds proceed at different paces, and a one-size-fits-all approach will inevitably fail to engage at least two kinds of students:

- those that are further dragging behind after each session, skipping most practical exercises and writing very little code;
- those that already know how to write some code and would appreciate a higher degree of complexity in the presented material.

An example of a first-year course with a strong practical component is CS1801 – Object-oriented programming. It spans over the two teaching terms and is taken by all single and joint-honours undergraduate computer science degrees. During the course, students learn the concepts of object-oriented programming that will be used through the entire degree and in their future profession. Acquiring those essential skills is critical to their success during the following years.

B. The CS1801 course

The course structure follows a traditional approach. Every week, there are two one-hour lectures, delivered to all the cohort in one go, and one three-hour practical session, organised in groups of approximately 30 students each. The uneven level of the students prevents the lecturer, during a practical session, from addressing the attendants as a whole. That problem is addressed by the presence of TAs for one-to-one interactions with students that have questions or are lost.

Each practical session is attended by three TAs, one per 10 students, which amounts to an interaction average time of 18 minutes per student per session. This can seem a lot of time, but in a three-hour session it corresponds to 6 minutes per hour – too short a time for detailed feedback.

Students are given a lab sheet that they are supposed to complete by the end of the session, but can carry from one session to the following ones, proceeding at their own pace. The lab sheets consist of several programming exercises, which the students have to code, compile and run. The exercises cover material from the lectures, and also include revisions from previous practical sessions. Completing the lab sheets plays an essential role in acquiring the skills of a proficient programmer: any developer knows that going from the pseudo-code written in a piece of paper to the real thing running on a computer requires a great deal of craftsmanship [2]. This learn-by-doing process of trial and error is regarded as one of the central aspects of education, and is not specific to computer science. For instance, In his 1984 book on *experiential learning* [8], Kolb emphasises the role of discovery and experience as sources of learning and development.

What we observe, however, is that many students fail to engage with the lab sheets: they do not tackle most of the exercises, leaving the sessions with important gaps in essential skills. This leads to a gradual block in their progress (most tasks require material from previous checkpoints). Furthermore, given that the acquisition of craftsmanship is a slow process requiring repetition, the more they drag behind, the less a chance they have to recover: it becomes a slow road to failing the course and, most probably, the first year.

The lack of engagement of some other students happens for different reasons: the presented material is trivial for their level; attending practical sessions is viewed as a mandatory boring activity. To address that issue, there are exercises, towards the end of the lab sheet, specifically aimed at highly-skilled students. They present interesting challenges requiring both problem solving and code proficiency. These are exercises that benefit from fruitful one-to-one interactions, which would need to be longer that the 18 minutes allocated to each student.

Overall, regardless of the motives, we observe a general tendency among students: even if the TAs are available and willing to help, most students will choose skipping exercises over calling the TAs.

III. An experiment in the practical sessions

A. Checkpoint system

During the last two academic years, with the main goal of overcoming the lack of engagement, but also in response to feedback provided by student and staff on the teaching of programming, we have conducted a formative assessment experiment in the CS1801 practical sessions: in each session, students have several checkpoints that help us (and them) keep track of their progress.

The process is coordinated by a lecturer, and follows the steps detailed below. We would like to note, as an early motivation of a blended model, that all these steps were conducted without any tool support, being therefore onerous in terms of time and dedication.

1) Lab sheet: A highly structured lab sheet is published on Moodle at the start of each practical session – this contains not only exercises, which are separated by well-identified (and strategically placed) checkpoints, but also examples and hints on how to approach the different tasks.

2) Practical session: During the session, each TA has a list with the checkpoints that every student has completed so far. When a student reaches a checkpoint and calls a TA, the checkpoint is verified – which consists in checking if the tasks were correctly completed, and the checkpoints list is updated accordingly.

3) Processing the information: Before the next practical session, the lecturer merges the information from the individual sheets, and publishes the updated list on Moodle. The history of those weekly updates is kept for reference – it shows the individual learning curves.

4) Publishing the results: At strategic points during the term, the lecturer produces a report with every student's achievements. This is used for internal monitoring of progression, and for deciding on any remedial actions.

B. Verifying the checkpoints

Every student has a different way of approaching checkpoints: some will regularly call the TAs at each checkpoint, others will prefer to have them verified in bulk. Nonetheless, whatever the approach, we have identified three stages at which students choose to have their checkpoints verified.

1) Complete solutions: Some students make sure that they have gone as far as they can, and present a program that not only works, but often will have different solutions for the same task. They expect the TA to give them advice on which solution is better, to comment on the code, and to encourage them to dig deeper.

2) Good attempts at a solution: Most students will consider a checkpoint ready to be verified as soon as they have written a snippet of code that compiles and presents a reasonable behaviour. Those programs will often miss particular cases and clever solutions, and the role of the TA is to inform the student about those alternatives.

3) Insufficient work: There are a few students that call the helper as soon as they have something that resembles the desired outcome. They expect to get major help from the TA in order to complete the task.

C. Acting on the results

One of the main aspects of using the checkpoint system is the ability to act on the gathered information. This is done at two levels: the information concerning a student allows for quick remedial actions; the information about the progression of the class as a whole provides a measure of the overall pace, informing the design of the lab sheets from session to session. If the entire class is lagging behind on a specific lab sheet, the following session can be lighter, allowing the students to catch up and relieve a possible frustration.

IV. Analysis of the experiment

A. Student engagement

The results of the experiment are presented in Table I, which only covers the first term of each year. There are significant differences in the approach to course delivery within both terms, which led us to focus mainly on the first term.

1) Differences between the two terms: In general, the level of student engagement drops during the second term, which is partially due to an increase in the number of checkpoints from one term to the other. Both terms have eleven practical sessions, but the last four sessions of term one are dedicated to the students that still have checkpoints to verify. All the other students no longer have to attend the sessions, dedicating this extra free time to other courses where their new skills will thrive in concrete applications (like, for example, games or robotics). For the students that are required to attend the remaining CS1801 sessions, there is no introduction of new material, and given the reduced number of students, the average support time from TAs is much higher.

We have observed that, during the second term of both years, the increase of 50% in the number of checkpoints (from 26 to 44 in 2014/15, and from 28 to 40 in 2015/15) leads to a decrease of approximately 75% in the overall rate of completion, when compared to the corresponding first term. The material studied in the practical sessions is also more complex during the second term, and the number of exercises is higher. All these contribute to the lack of engagement during the second term, and one of the reasons we went for the design of a blended model is to achieve, during the entire year, the success rate of term one.

TABLE I
STUDENT ENGAGEMENT DURING THE FIRST TERM

	2014/15	2015/16
Number of checkpoints	24	28
Number of students	83	113
Students with more than 50% of the tasks verified	93%	88%
Students with more than 75% of the tasks verified	87%	77%

TABLE II
CS1801 RESULTS FOR NON-REPEATING STUDENTS

	2012/13 – 2013/14	2014/15 – 2015/16
Grade average	59.4%	66.2%
Failing students	19.6%	13.6%

2) The success of term one: In term one, as Table I shows, approximately 90% of the students complete more than half of the tasks. Given that this corresponds to the least amount of work expected from an average student, the percentage shows a very good level of overall engagement. Moreover, approximately 80% of the students complete more than three quarters of the tasks, which shows a high level of participation.

These numbers are not directly comparable to any measures from previous years (no control was done during practical sessions), but a measurable outcome is the level of success in the mid-year CS1801 test, which is a piece of formative assessment conducted on a weekly basis, from weeks 7 to 11 of term one. The students that achieve a grade above 85% are released from the lectures for the rest of the term. Usually, by the end of term one, most students will have succeeded in passing the test, but we have observed that, with the introduction of the checkpoint system, students are reaching the passing grade earlier.

3) Summative assessment outcomes: Summative assessment for CS1801 consists of several small pieces of coursework (10% of the final grade) and an exam (90% of the final grade).

The checkpoint system was introduced in 2014/15 and has been running for the last two academic years. Table II presents the outcomes of those two years, comparing them to the two previous years. In order to better measure the effect of the experiment, we have only considered students taking the exam for the first time – repeating students may have undertaken more programming courses, which gives them a clear advantage. The results indicate a significant increase in the success of summative assessment: higher grades and fewer failing students. However, these first observations still need to validated by a proper statistical analysis, for which we are gathering further data.

B. Further improvements

When looking at the numbers, and whilst the experiment can be considered successful, there are still 20% of the students that are missing one quarter of the checkpoints, which is a significant number. To identify these students, we need a closer analysis of the process. The students have different levels of interaction with the TAs; the nature of those fundamental and time-consuming interactions reflects the level of the student, and raises different types of frustration.

1) Skilled students: Students with a prior knowledge of programming are very confident of their skills, and fail to realise that, although they have some programming abilities, there are specific requirements (for instance the style of the code and its readability by other programmers), which they are not familiar with. They regard the checkpoint system as a boring formality, and will skip it altogether. However, most of those students, when forced to check their work by an engaging TA, come to the conclusion that having skilled programmers reviewing the code is always a fruitful source of knowledge.

Skilled students tend to persist on recurrent mistakes and, in general, overestimate their abilities. It is not rare to find, in solutions presented by those more experienced students, many fragments of *bad code* that reveal misunderstanding and confusion. One such example is the following Java snippet (while being fully functional, it certainly leaves a bad impression on any trained programmer):

```
public static boolean negate(boolean a) {
  if (a == true) return false;
  else return true;
}
```

The student that produces such a piece of code is certain of having reached the goal of the task, but is missing important notions of programming. Those gaps in the acquired knowledge are mended by the interaction with TAs.

2) Struggling students: Students that are having a great deal of difficulty with programming feel uncomfortable asking for help, in particular when they look around and observe colleagues progressing at a faster pace than they are. When talking to those students, the variety of entry requirements becomes apparent: since neither mathematics nor programming are required, students that have taken those courses during their undergraduate studies are at an advantage. We also insist regularly on the fact that the learning outcomes should be measured by the end of the year, and that they should keep on trying. However, this does not prevent their frustration and/or impression of underachievement.

3) Limited resources: There is also the overhead of checking the exercises. The resources are limited, and sometimes students have to wait a considerable amount of time before getting checked. As they have to call a TA as soon one becomes free, this may end up being tiresome. Some students simply abandon the process midway.

C. Student satisfaction

There are several factors that affect the way students perceive the effectiveness of a course delivery. In the context of our experiment, we could identify two main perceptions among students: checkpoints are viewed either as a reward system, or as a remedial plan to address failure. Those different perceptions entail different ways of interacting with the system, leading to variable outcomes.

1) Feedback from students: Every term, students fill in a feedback questionnaire provided by Royal Holloway. Those questionnaires are highly detailed and present an overall picture of the course status. In the feedback forms for 2013/14 – term 1, 10 comments (in a total of 18) complain about the practical sessions. In the corresponding feedback forms from 2014/15, when the checkpoint system was introduced, 29 comments (in a total of 56) specifically mention good points about the practical sessions. The same tendencies were observed for the second term, although with less significant numbers. Anonymous comments included:

> "Compared to last year, the new format of the lab is much better."
> "The checkpoints in the labs were a good way of tracking progress."
> "Tasks on the lab sessions were chosen very well."

Besides those standard course feedback questionnaires, we foster further discussion with both staff and students to identify possible improvement opportunities. The yearly feedback provided by the students' committee for 2013/14 (the year preceding the introduction of checkpoints) mentioned that the practical sessions were delivered at a "too fast pace for people with little experience". During the next academic year, in the mid-term one-to-one meetings with their advisors, students were directly asked about the impact of checkpoints on their learning experience. The response was unanimous in acknowledging the system as extremely helpful. This impression was reiterated in the general feedback from the students' committee for 2014/15.

2) A rewarding system: The checkpoint system is perceived by students as a reward for their effort. Most students view this as an incentive, others as a way of monitoring their learning, and some even view the system as a competition. In general, they agree that the checkpoints really help them progress.

The responsible for the Student Experience in our Department has provided feedback about the system [9]: "The checkpoint system provides immediate formative feedback to the students. Students can identify problems and then in the lab discuss with the TAs what their problems are."

Given that a satisfied student is a better student, the positive feedback from students partially validates the effectiveness of the process. Furthermore, it also informs the approach, enabling a continuous fine-tuning of details in response to input from students.

3) Personal development: There is a growing awareness of the need to develop personal skills such as the ability to communicate, to present oneself with confidence, and to tackle unfamiliar problems. More generally, students' personal development should be understood to cover aspects of educational development (for example, the ability to use feedback to improve performance or to make use of educational resources such as Moodle or the library) and of career development (for example, understand professional issues associated with careers in IT or the ability to work effectively in teams).

Questionnaires give evidence of students' perception of the quality of support that they received. However, to act on such responses and improve our support, one also needs to:

- understand the expectations that students have in relation to their personal development;
- ensure that students understand how, through the received feedback, they are given the opportunity to develop those personal skills.

That is, students need to take ownership of their development, and they cannot do so on a vague understanding of what support they are provided with.

Our Head of Department has provided feedback about the system in this context [9]: "The checkpoint system is an excellent contribution to allowing students take more ownership of their personal development. It allows them to progress at their own pace by getting quick feedback on their performance and understand what they need to improve."

D. Informed teaching

At first, the system was introduced to promote students' engagement, by having them asking for help and interacting with the TAs. However, it ended up covering many other aspects of the student's learning process that deserve more attention and development. The system can, namely:

- maximise success and minimise failure in practical courses by continuously giving feedback to students on their progression;
- measure the pace at which each student progresses, allowing for early actions to be taken on the learning difficulties that are detected.

1) Acting on failure: The weekly feedback provided by checkpoints allows academics to take action early on.

A co-responsible for the CS1801 lectures has provided feedback about the system in this regard [9]: "The checkpoint system provides me with much needed timely feedback on student performance in the weekly labs. I can quickly grasp how well students keep up with the course material and can spot particularly challenging topics. This is invaluable especially in the first year, where many students do not easily come forward after lectures or during office hours to ask for clarification or help. In addition, it allows to spot students who seem to not be engaging with the course; their personal advisors can then focus their attention on these students during the tutorial sessions for the course."

Together with his feedback mentioned in Section IV-C2, the system has been deemed by the responsible for Student Experience, with respect to providing a continuous overall picture of students' progression, to provide staff members with feedback that is "immensely useful for the department particularly with respect to quickly identifying progress in CS1801 for the progressions committee."

2) Promoting success: During the first year of the experiment, we have regularly provided optional exercises in the lab sheets, aimed at the students that welcome harder challenges. However, we have observed that most students would simply ignore those extra tasks. We were even surprised by students complaining about not feeling challenged in the practical sessions, and confessing at the same time that they had never attempted to solve any of those extra tasks.

Under the hypothesis that this phenomenon was mainly due to the fact that those extra tasks did not correspond to an extra checkpoint, we have made small change to the lab sheets published during the second year of the experiment: for each practical session, we have moved those optional exercises into a *platinum* checkpoint. And, in every weekly status update, we have published the results of the platinum checkpoints along all the other ones. The result was surprising: this simple alteration has triggered a much higher student participation in those extra tasks.

Observing the success of platinum checkpoints, we are led to consider the system as a tool enabling a fine analysis of the students' behaviour, both when addressing failure and when promoting a higher level of engagement for highly skilled students.

V. Towards a blended model of education

The success of the experiment lies in both the achieved results and, more importantly, the room for improvement that the process seems to present with respect to automation. Indeed, there are several aspects of the checkpoint system that would greatly benefit from automated or semi-automated mechanisms involving online techniques.

A. Assessment and feedback

The checkpoint system provides continuous feedback on each student's progress. With an average of four checkpoints per practical session, students reach the end of term with more than 40 checkpoints.

Since the results are made available every week, both students and academics have a detailed perception of individual learning curves throughout the term. We aim to combine those qualitative appreciations with a score (1-3 stars) and a badge-reward system (similar to the ones implemented in games and other scenarios [10]). Platinum checkpoints are a first simple example of what special badges can achieve, namely when it comes to encouraging students to complete the most challenging exercises.

B. Bringing the MOOCs to the classroom

MOOCs have been, in the past two years, gaining traction as a model for massively teaching students off campus. Although the results are mixed and debatable, some of their characteristics are remarkable: active learning (self-pacing and instant feedback) and gamification (with, for instance, badge-awarding systems).

What we intend is to bring those successful aspects of MOOCs to the classroom. According to Anant Agarwal, CEO of edX, this process is about "taking ... the technologies we are developing in the large and applying them in the small to create a blended model of education" [11].

We propose to bring teaching, assessment and gamification techniques from MOOCs to the classroom, namely:

- students are able to follow the sessions at their own pace;
- progression to the next level only occurs when the previous levels have been completed, understood and verified;
- badges encourage students to perform better in the achievement of specific goals;
- the visualisation of progress bars and learning curves provides students with an overall perception of their performance.

C. Advantages of an automated system

We have identified the following possible improvements.

1) A first response line: If the code written by students is submitted to an IT system instead of being directly present to a TA, a few automatic tests will provide feedback to students on how to correct common mistakes. Several systems for automatic assessment of programming assignments exist, with different levels of support and feedback, as discussed in [12]. In our case the complexity of the tests can vary, but they constitute an automated line of action that will rely on human assistance only when necessary: the TAs will be called fewer times and their overall availability will increase. Furthermore, the automation trivially orders TA requests by submission time, which frees the student to start working on the next checkpoint after the submission of the code, instead of recurrently looking for an available TA.

2) Speeding up feedback: The automated submission also provides different levels of feedback, going beyond the typical interaction with the TAs. After passing the code through several tests, the student may get specific feedback on particular mistakes without interacting with a TA. Nonetheless, the TAs will always be available to provide further detail, if necessary. When the system is not able to provide automated feedback, a TA is called. Whatever the reason to call for human help, the responding TA will have a device with available information about the student's checkpoint history, the current checkpoint, and the code issues. This will speed up the process of interaction, during which the TAs can spend a considerable amount of time going through the checkpoints and looking for problems in the code before being able to help the student.

3) Increased promotion of different paces: The lab sheets are published on a weekly basis, rather then in bulk. The rationale behind this procedure is to allow the progressive unfold of a story during the term, for which the sessions sequence is essential. However, this approach has two main drawbacks:

- the faster students will finish the lab session in less than two hours, even when completing some optional harder tasks, which leaves out an hour that could be used to progress to the next tasks;
- some students will get stuck on a particular task and, although we insist that students may move on and come back later to that particular problem, some future tasks may depend on the left-behind task, and students get sometimes confused about what they know, and what they still have to acquire.

An automated system would overcome these issues by using a graph of task precedences that would allow all the tasks to be published at the same time, disabling those that require previous tasks to be completed. As soon as a student completes one checkpoint, the system will display any new available checkpoints. Also, any disabled checkpoint will display the tasks that are required for its unlocking. This also doubles as a knowledge map, where students can easily find paths leading to the acquisition of a specific skill.

D. Further aspects of an automated system

Beyond the advantages described in Section V, the checkpoint system automation would also provide a realtime source of information about the students' progress, both individually and as a whole. Different ways of looking at the data provide different insights. An automated system could easily provide charts with realtime statistical analysis, that can include the progression curve of one student or of the class, a classification of students according to several filter options (number of completed checkpoints, total score, ranking), comparison between several years and sessions, etc.

VI. Designing the application

In order to fulfil the mentioned goals and provide a better service to both students and academics, we have designed a web-based application that automates the checkpoints process. Running on the browser of any computer (desktop, laptop, tablet or phone), the application may be used by lecturers and students alike. The basic functionality of the checkpoints mechanism will be supported by a core module, and an API will provide the ability to extend the application with additional useful features that can enhance the learning experience of students, individually and as a whole. A proof-of-concept prototype has been developed and is functional, covering some of the essential features. The final application is under development since early 2016.

The outcome of this development is a software system that will not be specific to computer science, and that can be used by any discipline to provide formative assessment in practical sessions. The overall goals of the system are to:

- monitor student achievement and performance;
- reward the student with a score corresponding to those achievements and performances;
- measure the difficulty of exercises by looking at the overall performance of the class;
- adapt at runtime the delivery of the practical sessions to the level of the class, and propose course revisions for the next years;
- give tailored content to students that have specific difficulties or a higher level of achievement;
- provide a methodology for exercise-based sessions that can be used in a systematic way.

A. Overall description

During a typical session, students are working on computers where they can submit their checkpoints. TAs will have portable devices (tablets) that they use to identify students in need of help, and to verify the checkpoints.

When the users log into the system they have access to a general view of their account where they can either edit personal information, or select a course. Inside the course, each type of user has different options, as described below.

1) Students: Students are able to view rankings, individual achievements, and an overall picture of the checkpoints. In that view, students access the graph of checkpoints for the course, as well as the checkpoints that are completed and those that are unlocked, given the current score. There is also a graph with the learning curve of the student with respect to that course.

2) Academics: Academics are able to edit the graph of checkpoints, add new checkpoints, and monitor all the students' activities. They are also able to create rewards awarded to students when they reach certain points of the graph, or certain scores.

3) Administrators: Administrators have a different kind of access, allowing them to create courses and to perform functionalities such as management, monitoring and maintenance.

B. Definition of checkpoint

A checkpoint is a set of tasks that students have to perform. Usually that set is small, as fragmentation is an essential aspect of checkpoints: overcoming several small challenges is more rewarding than dealing with a complex task that will take long to complete [13].

The contents of a checkpoint can have several declinations: tutorial with simple tasks, complex tasks requiring problem solving, quizzes, etc.

The students' general view of all the checkpoints in the application highlights those that are accessible. Regardless of accessibility, each checkpoint displays a description of its content. This allows the student to understand, with the help of the graph, which checkpoints need to be completed before acquiring a specific knowledge.

The students' work on a checkpoint is independent of the software application, which only manages checkpoints and deals with the submitted material. This allows different kinds of disciplines to use the system. In the case of CS1801, for example, students work on a Linux server using an editor of their choice, and a Java compiler. Once the student considers that the tasks are completed, they upload the relevant files to the application and wait for the verification process.

C. Verifying a checkpoint

The solutions to the checkpoint tasks are verified by an automated system and/or a TA. The verification by an automated system requires the solution of the tasks to be submittable to an electronic system. In the absence of that feature, the system will always rely on a TA to verify a checkpoint. In this document, we will consider that the students are able to submit a file to get the checkpoint verified by an automated testing mechanism.

The verification process may have several outcomes, including the need to go back to the tasks in order to correct critical mistakes. One crucial aspect of a checkpoint is that is can be attempted several times: each attempt can result in a better outcome, which may also be awarded a higher score. This characteristic is aligned with literature on gamification [14], and with the article by Karpicke and Blunt [15] where they conclude:

> "Research on retrieval practice suggests a view of how the human mind works that differs from everyday intuition. Retrieval is not merely a readout of the knowledge stored in one's mind; the act of reconstructing knowledge itself enhances learning."

Hence, the students overcome mistakes by trial and error, which is a procedure they are familiar with [8].

Figure 1 shows the flow involved in the verification of a checkpoint. Each step of the process is detailed below.

1) Working on a checkpoint: The student is working on the tasks of the checkpoint. Some help from the TAs may be required, but students are encouraged to work on their own and submit solutions they deem adequate.

2) Submitting the tasks: When a student finishes a set of tasks, they upload the relevant files to the application, or fill in a form with the outcomes of the task (depending on the nature of the task). In the specific case of CS1801, the student submits a Java source file. As soon as the submission is done, the checkpoint becomes unavailable until either the verification is completed, or the process is cancelled by the student.

While waiting for the verification, the student may start working on any other unlocked checkpoint. The student may also choose not to submit the solutions to the tasks, but this means that no new checkpoints will ever be unlocked, and the student will eventually get stuck.

3) Testing the submission: A series of automated tests are performed. This first line of action looks for common mistakes that are easily detectable, and for which some feedback can be provided without any human intervention. Sophisticated tests can also be performed, if such a test suite is available for the discipline that is using the system. In the case of CS1801, a series of automated tests allow a complete verification of the checkpoint (see Section VI-D).

4) Getting feedback: If the tests are not passed, the system will try to provide some automated feedback to the student. If that feedback cannot be produced, the application puts the submission in the TAs' queue: a TA will be notified and the student will eventually be approached to get some oral feedback, or to have the checkpoint validated. This may happen when, for instance, the student has an easy-to-fix error in the code.

5) Validating a checkpoint: The validation of a checkpoint can be automatically completed by the application, or require the intervention of a TA. In the latter case, the TAs use their tablets to capture a QR code on the computer of the student (uniquely identifying the student-checkpoint pair).

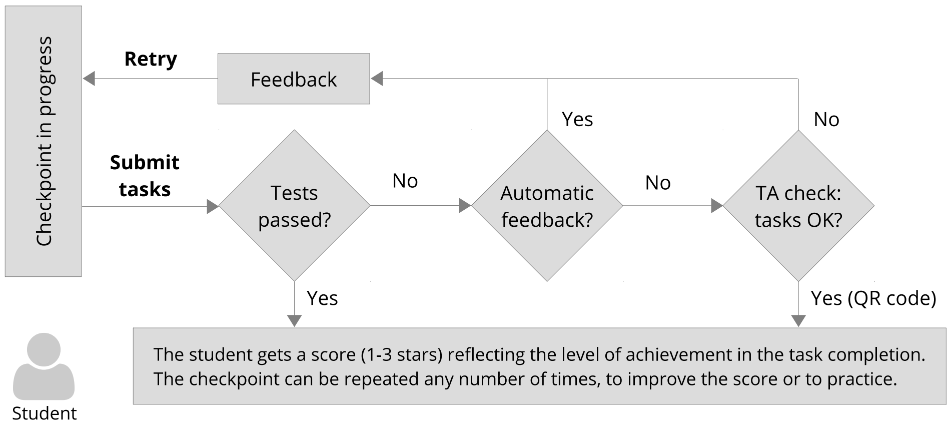
Checkpoint in progress
Retry
Feedback
Submit tasks
Tests passed?
No
Automatic feedback?
Yes
No
TA check: tasks OK?
No
Yes
Yes (QR code)
The student gets a score (1-3 stars) reflecting the level of achievement in the task completion.
The checkpoint can be repeated any number of times, to improve the score or to practice.
Student

Fig. 1. Verifying a checkpoint with the application

This will bring up an interface that allows the TA to score the checkpoint. When the checkpoint gets validated, some new checkpoints may become unlocked.

6) Score: When the checkpoint is validated, the student gets a score (1-3 stars) based on several criteria. This score can be automatically provided by the system, according to the outcomes of the test suite, or it can be attributed by the TA.

D. The submission assessment for CS1801

The submission of a checkpoint consists of uploading the file with the code to the server. The file is then submitted to a series of tests that check if the code corresponds to what was asked in the set of tasks. The tests are sequential and follow the steps described below, shown in Figure 2.

1) Compilation: The first requirement to complete a checkpoint is that the submitted code compiles. If the compilation issues warnings, the student may be given some automated feedback that will suggest improvements on the code.

2) Unit testing: The exercises have specific outcomes that allow unit testing. For example, a program that defines a class can be tested by creating an object of that class and calling its methods. There are several tests for each exercise; those tests can be weighted in order to assess the student accordingly. The number of stars the student is awarded for the checkpoint will increase with the number of passed tests.

3) Checking the style: Students often write code that does the required job, but presents stylistics flaws that, even if they are optimised by the compiler, show some gaps in the acquired knowledge. One of the goals of CS1801 is to get students to write programs that are readable and free from *bad code*.

E. Prototype

We have developed an early prototype that covers the following functionality:

- log in with testing accounts;
- set different groups of checkpoints (emulating a course);
- organise, in each group, a linear sequence of checkpoints;
- view the stages of completion of each group of checkpoints;
- view the unlocked checkpoints;
- view the locked checkpoints, and the precedences that unlock them;
- verify a checkpoint using a QR code.

F. Extensions to the prototype

The application extends the prototype with the following additional features, most of them via an API.

1) Remote authentication: The users can use their usual credentials, which are securely fetched from a remote server.

2) Timed checkpoints: When a student starts a set of tasks, a timer is activated to measure the time spent on that checkpoint.

3) Graphical representations of data: Several views of the students' progress, individual and by groups, will help both students and teachers have snapshots of the students' status and the learning curves.

4) Graph of checkpoints: Students can progress according to their actual skills, in a non-linear fashion, instead of having to follow a sequence of checkpoints that may not be the most adequate to their learning curve.

5) Automated verification process: The submissions are subjected to an automated suite of tests.

6) Enhanced gamification: Student's achievements will be awarded trophies, rewarding effort – not just success; those trophies are viewed by other students and establish peer motivation.

7) Students' feedback on the exercises: Academic staff will receive a constant measurement of how students perceive the exercises, allowing them take early action and adapt the exercises to the needs of the students.

8) Look and feel: The new features will require a revision of the application user interface.

VII. Related Work

A. Checkpoints

The notion of checkpoints applied to support teaching of programming has been used for some time. The concrete goals have ranged from formal assessment [16] to testing the rate of student progress and improve the process [17]. The

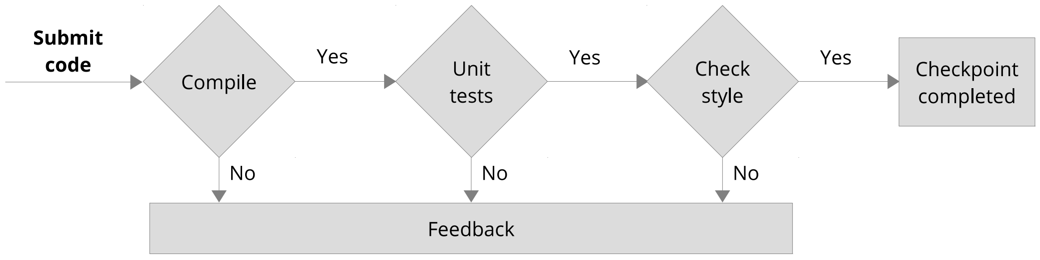

Fig. 2. Automated test sequence for CS1801

granularity at which these are used also varies significantly. In some models, checkpoints are set after small exercises are completed [18], whilst others set these only when more considerable tasks are finalised [19]. In the case of our approach, the checkpoints are designed at a low level of granularity to allow both a detailed view of student progress, and a good understanding of the difficulty in each set of tasks. However, the application is also suited to different approaches, like for instance long sessions focusing on one single difficult task.

The way the completion of these checkpoints is taken into account also varies. The results are logged in various fashions, ranging from simple manual accounting to free [20] and commercial tool support [21], [22]. The model we present includes the design of a toolchain that takes into account the required granularity of the specific checkpoints, fosters a good user experience for both assessors and students (e.g. the use of QR code recognition so that manual input is minimised), allows for a graph of dependencies, and is integrated with some gamification concepts.

Checkpoints are also used either as a means to assist in supporting students in traditional programming laboratory settings [21] or as part of automated assessment [20]. Our approach still includes traditional sessions, where students are presented with exercises and benefit from the support of teaching assistants. But it also takes advantage of modern techniques, such as those widely used in Continuous Integration, to reduce the amount of required intervention and to give immediate feedback to students.

B. Serious games and gamification

Serious games were introduced in order to facilitate purposes other than simply entertainment. These ranged from games in a general sense as described by Abt in 1970 [23], to digital games as presented more recently by Sawyer and Rejeski [24] and Michael and Chen [25]. Whilst sharing some concepts with serious games, *gamification* [7] focuses on using game elements to enhance several activities, rather than having users specifically play a game. These activies include learning [26], [27], business [28], and even self-help [29]. The model we describe uses the concept of gamification to improve student engagement in practical sessions (and with the goal of improving attainment), whilst resulting in better learner analytics.

C. Blended learning

Blended learning [3], [4], corresponds to models where the Internet and digital media are combined with traditional classroom settings. In his 2012 report [30], Friesen presents the following definition:

> "Blended learning" designates the range of possibilities presented by combining Internet and digital media with established classroom forms that require the physical co-presence of teacher and students.

This technique was used in the context of our approach as a means to reap benefits from both worlds. Traditional practical sessions allow close support and quick and helpful feedback, whilst the digital content and automatic checks accelerate the administrative process and allow students to focus in learning how to think computationally.

D. Automatic checks

Automatically checking student programming work has been researched for many years [31], [32], and is still an active area [12], [33]. Whilst it is possible to use these techniques to address, for instance, issues of scale, the level of feedback and assistance still lags behind what can be achieved with direct support from experienced teaching assistants. Notwithstanding this gap, it is a useful approach that can be integrated with the traditional one. In the model we describe, this has been achieved by including automated checks to address a first line of issues, followed by human support when required.

VIII. Conclusion

We have presented a blended model of education motivated by a successful formative assessment experiment conducted in the practical sessions of a first-year undergraduate programming course. That model will benefit from the support of an application, which design was informed by building and testing a functional prototype. The application is currently under development, and will be deployed in 2016/17.

One of the main improvements in the final application, compared to the prototype, is the strong gamification component. We have observed how the introduction of platinum checkpoints has motivated some students to engage with harder tasks. We expect, similarly, the star-based scores and the achievement badges to stimulate a higher level of overall engagement. Moreover, given the current trends in gamification research, this is an aspect with much room for improvement: the API will easily accommodate third-party awards like, for instance, Mozilla Open Badges [10].

We will evaluate and monitor the initiative mainly through: student feedback (advisor tutorials, one-to-one meetings and student feedback questionnaires); results of further formative and summative tests, including the final CS1801 examination.

This will allow us to consider any necessary improvements and evolve the model accordingly, before moving to a wider dissemination of the application that, by design, is not specifically targeted at teaching programming. Nonetheless, the system has the potential of creating a baseline of tools and techniques for other courses in our department that include exercise-based practical sessions, providing a clear added value: it allows the identification of students needing additional support, and measures the effectiveness/difficulty of particular tasks. By increasingly adopting this technique, the Department of Computer Science, as a whole, can have a better and up-to-date perception of student achievement and, hence, proactively make changes as necessary.

We also plan, in the future, to introduce a mechanism through which students can give a simple quantitative feedback on the quality and perceived difficulty of each set of tasks. Based on those pieces of information – as measured through performance and as perceived by students – academics will move yet another step towards taking more informed pedagogical decisions.

REFERENCES

[1] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year cs students," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '01. New York, NY, USA: ACM, 2001. doi: 10.1145/572133.572137 pp. 125–180.

[2] T. Jenkins, "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, 2002, pp. 53–58.

[3] D. R. Garrison and H. Kanuka, "Blended learning: Uncovering its transformative potential in higher education," *The internet and higher education*, vol. 7, no. 2, pp. 95–105, 2004. doi: 10.1016/j.iheduc.2004.02.001

[4] C. J. Bonk and C. R. Graham, *The handbook of blended learning: Global perspectives, local designs*. John Wiley & Sons, 2012. ISBN 9781118429570

[5] F. A. Marco, V. M. R. Penichet, and J. A. G. Lázaro, "Drawer: an innovative teaching method for blended learning," in *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems*, M. P. M. Ganzha, L. Maciaszek, Ed. IEEE, 2013, pp. 727–734.

[6] R. Murphy, L. Gallagher, A. E. Krumm, J. Mislevy, and A. Hafter, "Research on the use of Khan Academy in schools: Research brief," *SRI International*, 2014.

[7] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining "gamification"," in *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, ser. MindTrek '11. New York, NY, USA: ACM, 2011. doi: 10.1145/2181037.2181040 pp. 9–15.

[8] D. A. Kolb, *Experiential Learning: experience as the source of learning and development*. Prentice Hall, 1984. ISBN 9780132952613

[9] N. Barreiro and C. Matos, "Checkpoint system documentation," Department of Computer Science, Royal Holloway, University of London, Tech. Rep., 2016, internal document.

[10] "Mozilla Open Badges," accessed: 2016-05-09. [Online]. Available: http://openbadges.org/

[11] A. Agarwal, "Why MOOCs (still) matter," 2013, TED talk by Anant Agarwal, CEO of edX, Accessed: 2016-05-06. [Online]. Available: http://bit.ly/1kZwi9Y

[12] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '10. New York, NY, USA: ACM, 2010. doi: 10.1145/1930464.1930480. ISBN 978-1-4503-0520-4 pp. 86–93.

[13] T. Amabile and S. Kramer, *The progress principle: Using small wins to ignite joy, engagement, and creativity at work*. Harvard Business Press, 2011. ISBN 9781422198575

[14] J. McGonigal, *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. Penguin Group , The, 2011. ISBN 9780143120612

[15] J. D. Karpicke and J. R. Blunt, "Retrieval practice produces more learning than elaborative studying with concept mapping," *Science*, vol. 331, no. 6018, pp. 772–775, 2011. doi: 10.1126/science.1199327

[16] J. Bennedsen and M. E. Caspersen, "Assessing process and product: a practical lab exam for an introductory programming course," *Innovation in Teaching and Learning in Information and Computer Sciences*, vol. 6, no. 4, pp. 183–202, 2007. doi: 10.11120/ital.2007.06040183

[17] S. Cvetkovic, R. Seebold, K. Bateson, and V. Okretic, "CAL programs developed in advanced programming environments for teaching electrical engineering," *IEEE Transactions on Education*, vol. 37, no. 2, pp. 221–227, 1994. doi: 10.1109/13.284998

[18] University of Edinburgh, "Physics 2A Scientific Programming in JAVA," 2009, course notes, Accessed: 2016-05-07. [Online]. Available: http://www2.ph.ed.ac.uk/~aturner/teaching/Scientific-Programming/documentation/booklet.pdf

[19] D. Parsons and P. Haden, "Programming osmosis: Knowledge transfer from imperative to visual programming environments," in *Conference of the National Advisory Committee on Computing Qualifications*, 2007.

[20] S. Zhigang, S. Xiaohong, Z. Ning, and C. Yanyu, "Moodle plugins for highly efficient programming courses," in *1st Moodle Research Conference*, 2012.

[21] E. Seung, "Examining the development of knowledge for teaching a novel introductory physics curriculum," Ph.D. dissertation, Purdue University, 2007.

[22] "WebAssign," accessed: 2016-05-07. [Online]. Available: http://webassign.net/

[23] C. C. Abt, *Serious games*. Viking Press, 1970. ISBN 9780670003136

[24] B. Sawyer and D. Rejeski, "Serious games: Improving public policy through game-based learning and simulation," 2002.

[25] D. R. Michael and S. L. Chen, *Serious Games: Games That Educate, Train, and Inform*. Muska & Lipman/Premier-Trade, 2005.

[26] K. M. Kapp, *The gamification of learning and instruction: game-based methods and strategies for training and education*. John Wiley & Sons, 2012. ISBN 9781118096345

[27] B. Kumar and P. Khurana, "Gamification in education-learn computer programming with fun," *International Journal of Computers and Distributed Systems*, vol. 2, no. 1, pp. 46–53, 2012.

[28] B. Burke, "Gamification: Engagement strategies for business and IT," Gartner Inc, Tech. Rep. G00245563, 2012.

[29] A. M. Roepke, S. R. Jaffee, O. M. Riffle, J. McGonigal, R. Broome, and B. Maxwell, "Randomized controlled trial of SuperBetter, a smartphone-based/internet-based self-help tool to reduce depressive symptoms," *Games for health journal*, vol. 4, no. 3, pp. 235–246, 2015. doi: 10.1089/g4h.2014.0046

[30] N. Friesen, "Report: Defining blended learning," 2012, accessed: 2016-05-08. [Online]. Available: http://learningspaces.org/papers/Defining_Blended_Learning_NF.pdf

[31] J. Hollingsworth, "Automatic graders for programming classes," *Communications of the ACM*, vol. 3, no. 10, pp. 528–529, Oct. 1960. doi: 10.1145/367415.367422

[32] J. B. Hext and J. W. Winings, "An automatic grading scheme for simple programming exercises," *Communications of the ACM*, vol. 12, no. 5, pp. 272–275, May 1969. doi: 10.1145/362946.362981

[33] N. A. Rashid, L. W. Lim, O. S. Eng, T. H. Ping, Z. Zainol, and O. Majid, *Advanced Computer and Communication Engineering Technology: Proceedings of ICOCOE 2015*. Springer International Publishing, 2016, ch. A Framework of an Automatic Assessment System for Learning Programming, pp. 967–977.