A Racket-Based Robot to Teach First-Year Computer Science

K.Androutsopoulos, N. Gorogiannis, M. Loomes, M. Margolis,
G. Primiero, F. Raimondi, P. Varsani, N. Weldin, A.Zivanovic
School of Science and Technology
Middlesex University
London, UK
{K.Androutsopoulos|N.Gkorogiannis|M.Loomes|M.Margolis|G.Primiero|F.Raimondi|P.Varsani|N.Weldin|A.Zivanovic}@mdx.ac.uk

ABSTRACT

A novel approach to teaching Computer Science has been developed for the academic year 2013/14 at Middlesex University, UK. The whole first year is taught in an holistic fashion, with programming at the core, using a number of practical projects to support learning and inspire the students. The Lisp derivative, Racket, has been chosen as the main programming language for the year. An important feature of the approach is the use of physical computing so that the students are not always working "through the screen", but can experience physical manifestations of behaviours resulting from programs. In this paper we describe the MIddlesex Robotic plaTfOrm (MIRTO), an open-source platform built using Raspberry Pi, Arduino, and with Racket as the core coordination mechanism. We describe the architecture of the platform and how it can be used to support teaching of core Computer Science topics, we describe our teaching and assessment strategies, we present students' projects and we provide a preliminary evaluation of our approach.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms

Theory,Human Factors.

Keywords

Educational approaches and perspectives, Experience reports and case studies

1. INTRODUCTION

Designing a new undergraduate programme requires a number of choices to be made: what programming language should we teach? Which development environments? Should mathematical foundations play a dominant role, or will they discourage students from attending? Moreover, the current stand of our educational system with respect to industry seems to rely on a discouraging contradiction: on the one hand, it is tempting to market new undergraduate programmes with the claim that they will provide the skills required by industry. On the other hand, we argue that the only certainty is that students will live in a *continuously evolving environment* when they leave education, and that it is not possible to forecast market requests in a few years' time.

In the design of a new Computer Science programme for the academic year 2013/2014 we have been driven by the requirement that we should prepare students for change, and that we should teach them how to *learn new skills* autonomously. Students entering academia may not be prepared for this: they could be arriving from high school where the focus is on achieving good grades in specific tests. How do we achieve the objective of preparing good *learners*?

We decided to employ the Lisp-derivative Racket to support the delivery of a solid mathematical background and the creation of language-independent programming skills. Moreover, we decided to work on *real hardware* so that the students could appreciate the result of executed code. The work is organised around projects involving Arduino, Raspberry Pi, and a Robot that we describe here.

We have completely revised our delivery and assessment methods to support our aims. There are no modules or courses and the activities run seamlessly across the projects. The assessment method is not based on exams, but on *Student Observable Behaviours* (SOBs), that are fine-grained decompositions of learning outcomes providing evidence of students' progress.

Many of the elements in this approach have been tried elsewhere, including problem-based learning, assessment through profiling and using Lisp as a first programming language. We believe, however, that this programme takes these ideas further than previously, and also blends these in ways that are unique. The integration of Lisp (Scheme) and formalisms in an holistic way was introduced at Hertfordshire by one of the authors many years ago [8], but only in the context of a single module. Several years earlier, a highly integrated curriculum was designed in a project funded by a large company in the UK, to develop formal methods in software engineering practice [7], but this was for small cohorts of students at Master level. From a pedagogical viewpoint, our approach broadly recalls a fine-grained outcome-based

European LISP Symposium 2014 Paris, France

learning path model, but the theoretical implications remain to be assessed in their full meaning, especially for the pedagogical support (see [14] for a recent overview). Finally, an essential aspect of our course structure is the integration of the Lisp-based programming methodology with a range of issues in electrical engineering, robotics and web-based applications. While other educational programmes have often preferred to drop Lisp variants in favour of other more dedicated programming environments (e.g. in the famous case of MIT 6.001 course based on Scheme and [1] redesigned with Python for Robotics applications) we intend to preserve the more in-depth and foundational understanding of programming that a Lisp-style language can offer and at the same time offer a greater flexibility with respect to real-world challenges.

In this paper we focus on how Racket has provided a solid support for our new strategy: in Section 2 we describe the overall structure of the first year and the progress of students from simple examples to more complex scenarios; this progress enables the students to control a real robot, described in Section 3. In section 4 we describe our assessment strategy and we present a tool to support it. An evaluation of our approach is provided in Section 5, where we describe students' projects and various measures for engagement, attendance and overall progress.

2. OVERVIEW OF THE FIRST YEAR

In our new first year of Computer Science, there are no modules or courses and all the activities run across various sessions during the week. The idea is that employing a problem-driven approach, we give students the confidence needed to study independently. In essence, this is our way to teach them *"how to learn"*.

Each week consists of the following structured sessions: *lecture*, *design workshop*, *programming workshop*, *physical computing workshop*, *synoptic workshop*.

- **General Lecture.** A two-hour lecture is given, introducing or developing a topic and related projects. However, this is not where learning should happen: we envisage our lectures as motivational and high-level descriptions of the activities that will follow during the week.

- **Design Workshop.** In these workshops students develop skills required to work in a design environment. Design might be built in software (programming) or hardware, it might involve bolting existing systems together (systems engineering), or developing processes for people who are using the systems (HCI). We cover ways of generating ideas, ways of representing designs so that they can be discussed, professional ways of criticising designs and ways teams of people work together to produce and deliver designs. Delivery happens in an open-space flexible environment, with large tables that can be moved around and arranged in small groups, and the workshop lasts two hours. Students may be asked to present in front of the class the result of their work.

- **Programming Workshop.** In the two-hour programming workshops we help the students with exercises, master-classes, coaching sessions, to develop their fluency in coding. We have restricted the first year to looking at just one main language, Racket [11], a functional language derived from LISP. Racket should be new to most students, thus ensuring that the students are all at the same level of experience so that we can focus on teaching best practises rather than undoing bad habits. The choice of a programming language was one of the most carefully debated issues in the design of this new course. Racket was selected for the availability of a number of libraries that support teaching, for its integrated environment (DrRacket) that allows obtaining results with very minimal set-up, and for the availability of a large number of extensions including libraries to interact with networking applications such as Twitter, libraries for Arduino integration and environments for graphics, music and live-coding.

- **Physical Computing Workshop.** The output of software systems increasingly results in tangible actions in the real world. It is very likely that the most common piece of software students will see in their jobs is not a relational database to store sales, but a procedure to manage self-driving cars. As a result, we think that students should be exposed to a wide variety of physical devices that are crucial to understanding computer science. These will range from simple logic gates (the building blocks of every computer currently commercially available), to microcontrollers (Arduino) and other specialist devices. The emphasis is on programming using Racket, not building, these devices. In this two-hour workshop we also explore how to interface these, and how people interact with computers using such devices.

- **Synoptic Workshop.** This is where we "pull everything together" by taking multiple strands of activity and fit all of the bits together. It is longer than the other workshops (4 hours) to allow time to design, build, test and discuss projects. This is not simply about 'applying' what has been learnt - it is about learning and extending what is known in a larger context.

In each of the Programming, Physical and Synoptic Workshops, one staff member and two Graduate Teaching Assistants attend to around 20 students. In the Design session the number of students rises to 40. Students do most of their study during class hours, but handouts contain exercises for self-study and they have almost continuous access to the laboratories to work independently on physical computing.

2.1 Growing Racket skills

Our delivery of Racket starts with the aim of supporting the development of a traffic light system built using Arduino boards [2, 9], LEDs and input switches. The final result should be a system with three traffic lights to control a temporary road-work area where cars are only allowed in alternate one-way flow and with a pedestrian crossing with request button.

Arduino is a microcontroller that can run a specific code or can be driven using a protocol called Firmata [3]. We employ this second approach to control Arduino boards from a different machine. To this end, we have extended the Firmata Racket library available on PLaneT [13] to support Windows platforms, to automatically recognise the USB/serial port employed for connection and to support additional kinds of

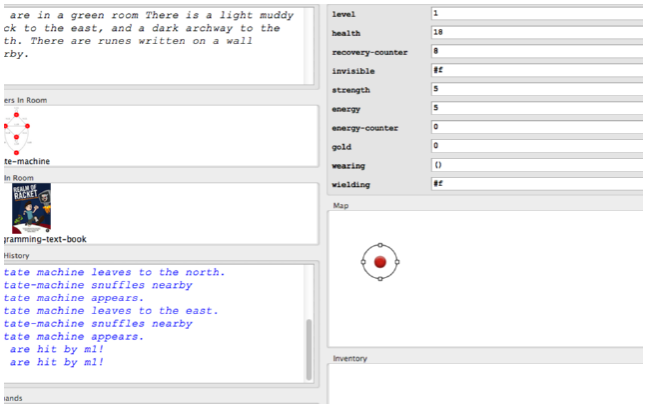

Figure 1: A screenshot of the Dungeon Game Interface

messages for analog output and for controlling a robot (see next section). Our library is available from [12].

Students employ this library in the first week to start interacting with DrRacket using simple code such as the following:

```
#lang racket
(require "firmata.rkt")
(open-firmata)
(set-pin-mode! 13 OUTPUT_MODE)
(set-arduino-pin! 13)
(sleep 1)
(clear-arduino-pin! 13)
```

This code turns an LED on for a second and then turns it off. Students then start working on lists and see traffic lights as lists of LEDs. High order functions are introduced to perform actions on lists of LEDs, such as in the following code that sets Arduino PINs 7, 8 and 9 to OUTPUT mode:

```
#lang racket
(require "firmata.rkt")
(open-firmata)
(define pins '(7 8 9))
(map (lambda (pin)
       (set-pin-mode! pin OUTPUT_MODE))
     pins)
```

As part of this project students learn how to control events in a timed loop using *clocks* and by making use of the Racket function `(current-inexact-milliseconds)`. This also enables students to *read* the values of input switches and to modify the control loop accordingly.

The result of this project is typically approximately 200 to 500 lines of Racket code with simple data structures, high order functions and the implementation of control loops using clocks.

Following this Arduino project, students explore a number of other Racket applications, including:

- A dungeon game with a GUI to learn Racket data structures. See Figure 1.
- The Racket OAuth library to interact with the Twitter API. A Racket bot is currently running at `https://twitter.com/mdxracket`, posting daily weather forecast for London. A description of this bot is available at `http://jura.mdx.ac.uk/mdxracket/index.php/Racket_and_the_Twitter_API`.
- A Racket web server to control an Arduino board. More details about this are available at `http://www.rmnd.net/wp-content/uploads/2014/02/w2-programming.pdf` (this is the handout given to students for their programming and physical computing workshop in one week).

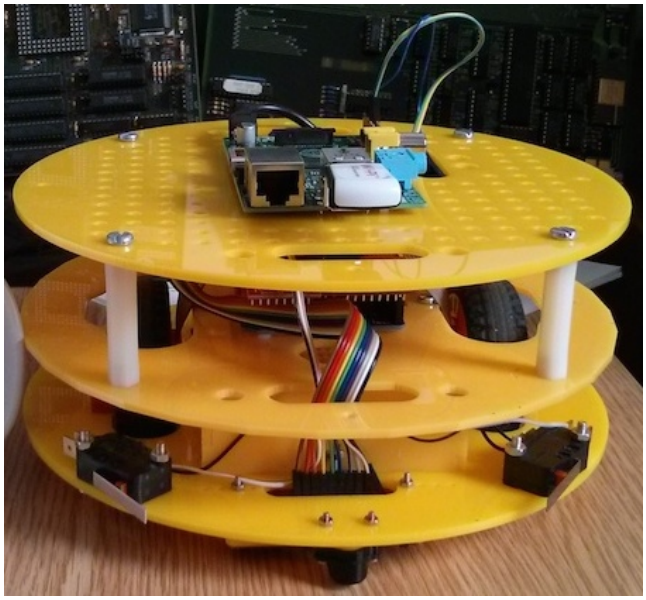

Figure 2: The Middlesex Robotic Platform

All these elements contribute towards the final project: develop Racket applications for the Middlesex Robotic Platform (MIRTO), described in the next section.

3. MIRTO ARCHITECTURE

The MIddlesex Robotic plaTfOrm (MIRTO, also known as Myrtle), shown in Figure 2, has been developed as an open-source platform that can be used across different courses; its current design and all the source code are available online [10]. The Middlesex Robotic platform shown is composed of two units (from bottom to top):

1. The base platform provides wheels, power, basic sensing and low level control. It has two HUB-ee wheels [4], which include motors and encoders (to measure actual rotation) built in, front and rear castors, two bump sensors and an array of six infra-red sensors (mounted under the base), a rechargeable battery pack, which is enough to cover a full day of teaching (8 hours) and an Arduino microcontroller board with shield to interface to all of these. An extended version of Firmata (to read the wheel encoders) is running on the Arduino, which provides a convenient interface for Racket code to control and monitor the robot.
2. The top layer (the panel on top in Figure 2) is where higher level functions are run in Racket and consists of a Raspberry Pi, which is connected to the the Arduino by the serial port available on its interface connection. The Raspberry Pi is running a bespoke Linux

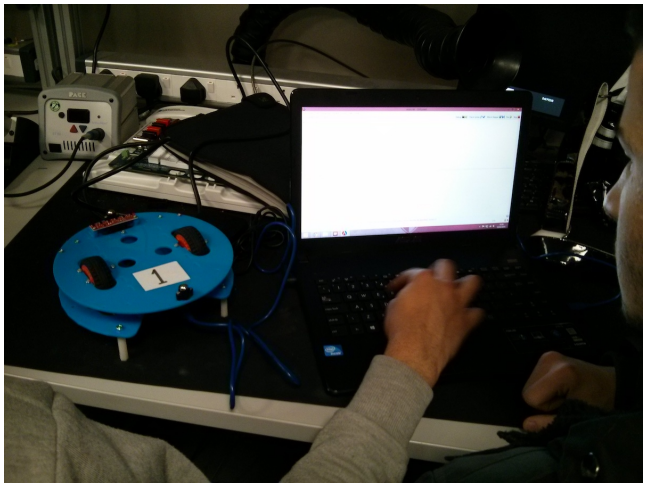

Figure 3: MIRTO Arduino layer connected directly to a PC

image that extends the standard Raspbian image; it includes Racket (current version 5.93), and is using a USB WiFi adapter to enable remote connections via SSH and general network activities. This layer enabled us to also use cameras, microphones and text to speech with speakers to extend the range of activities available to students. Additional layers can be added to the modular design to extend the robots capabilities.

The robotic platform is certainly a helpful artifact to engage students more, but it also represents a way to combine our crucial interest in the formal and theoretical aspects underlying computing. In fact, students start using the robot to investigate product of finite state machines (computing the product of the state space of the two wheels) and continue studying all the relevant formal properties that they see implemented on MIRTO. They then move to connecting the Arduino layer directly to a PC, see Figure 3. We have built a bespoke Racket module for this interaction (see Section 3.1); from the students' point of view, this is essentially a step forward with respect to a "simple" traffic light system, and they can re-use the control loops techniques employed for the first project to interact with wheels and sensors. After getting familiar with this library, students progress to study networking and operating systems concepts: this allows the introduction of the top layer, the Raspberry Pi. Students can now transfer their code from a PC to the Raspberry Pi and they control MIRTO over a wireless connection. This allows the introduction of control theory to follow a line and other algorithms (such as maze solving). We present some details of the code in the following section.

3.1 A Racket library for MIRTO

We have built a Racket library for MIRTO that allows students to interact with the robot by abstracting away from the actual messages exchanged at the Firmata level (see the file `MIRTOlib.rkt` available from [10]). The library provides the following functions:

- `setup` is used to initialise the connection between a Racket program and the Arduino layer (this function initialises Firmata and performs some initial set-up for counters). Correspondingly, `shutdown` closes the connection.
- `w1-stopMotor` and `w2-stopMotor` stop, respectively, the left and the right wheel. The function `stopMotors` stop both wheels.
- `(setMotor wheel power)` sets `wheel` (either 1 or 2) to a certain power, where `power` ranges between -100 (clockwise full power) and +100 (anti-clockwise full power). `(setMotors power1 power2)` sets both motors with one instruction.
- `(getCount num)` (where `num` $\in \{1, 2\}$) returns the "count" for a wheel. This is an integer counter that increases with the rotation of the wheel. A full rotation corresponds to an increase of 64 units for this counter. Given that the wheel has a diameter of 60 mm, it is thus possible to compute the distance travelled by each wheel.
- `enableIR` enables infra-red sensors (these are initialised in an "off" state to save battery); `(getIR num)` (where `num` $\in \{1, 2, 3\}$) returns the value of the infra-red sensor. This is a number between 0 (white, perfectly reflecting surface) and 2000 (black, perfectly absorbing surface).
- `leftBump?` and `rightBump?` are Boolean functions returning true (resp. false) when a bump sensor is pressed (resp. not pressed).

The following is the first exercise that students are asked to do to move the wheels for one second:

```
#lang racket
(require "MIRTOlib.rkt")
(define (simpleTest)
  (setup)
  (setMotors 75 75)
  (sleep 1)
  (stopMotors)
  (shutdown))
```

This code moves the wheels for one second and then stops them. Students test this code using the Arduino layer only, as shown in Figure 3. Similarly to the traffic light project, students then move to more complex control loops and start using the Raspberry Pi layer using SSH and command-line Racket. The following snippet of code extracted from a control loop prints the values of the infra-red sensors every two seconds:

```
;; [...]
  (set! currentTime (current-inexact-milliseconds))
;;
  (cond ( (> (- currentTime previousTime) 2000)
          (map (lambda (i)
                 (printf " IR sensor ~a -> ~a\n" i
                 (getIR i)))
            '(1 2 3))
          (set! previousTime
              (current-inexact-milliseconds))))
;; [...]
```

The functions provided by the library allow the implementation of a Racket-based PID controller [6] for MIRTO. Students are also introduced to maze solving algorithms, which can be implemented using the infra-red sensors and the bump sensors. The Racket code for both programs is available from [10] in the `servos-and-distance` branch.

After these exercises and guided projects, students are asked to develop an independent project. We report some of these projects in Section 5.

4. ASSESSMENT STRATEGY

As mentioned above, the delivery of the first year of Computer Science has been substantially modified, modules have been removed and students are exposed to a range of activities that contribute to *projects*.

As a result, we have introduced a new assessment strategy to check that students have understood and masters the basic concepts required during the second year and are able to demonstrate these through practical demonstration. We use the term **Student Observable Behaviours** (SOBs) to refer to fine-grained decompositions of learning outcomes that provide the evidence that the students are progressing. Passing the year involves demonstrating SOBs. There are three types of SOBs:

1. **Threshold level SOBs** are those that *must* be observed in order to progress and pass the year. Students must pass *all* of these; a continuous monitoring of the progress using the tool described below ensures that any student who is at risk of not doing so is offered extra support to meet this level.

2. **Typical level SOBs** represent what we would expect a typical student to achieve in the first year to obtain a good honours degree. Monitoring this level provides a very detailed account of how each student is meeting expectations. Students are supported in their weak areas, encouraged not to hide them and not to focus only on the things they can do well. Our aspiration is to get the majority of students to complete all the typical level SOBs.

3. **Excellent level SOBs** identify outstanding achievements. These are used to present real challenges of different types to students who have demonstrated to be ready for them.

Projects have been designed to offer assessment opportunities both en-route and in the final project delivery. Projects are posed in such a way as to ensure that students who engage with the process have the opportunity to demonstrate threshold level SOBs. As a result, "failure" to successfully complete a project does not lead to failure to complete the threshold SOBs. Projects have a well-defined set of core ideas and techniques (threshold), with suggestions for enhancements (typical), and open-ended questions (excellent). Note that there is no concept of averaging or summation: in theory a student could complete all of the excellent level SOBs, but fail the year as a consequence of not meeting one threshold SOB. This is virtually impossible in practice, as staff are aware that there are outstanding threshold SOBs, and take the opportunity of observing them en-route. Of course, if a student really can't do something that has been judged threshold, we will deem it a failure.

Students who fail to demonstrate all threshold SOBs by the end of the academic year will, at the discretion of the Examination Board and within the University Regulations, be provided with a subsequent demonstration opportunity. This will normally be over the Summer in the same academic year. Resources including labs and support staff will be made available during this period.

The process of assessment and feedback is thus continuous via a "profiling" method. This method allows us to track every student in detail, to ensure that we are supporting development and progression. This means we have comprehensive feedback to the teaching team available in real time. Also, students have a detailed mechanism available to monitor their own progress. This includes ways of viewing their position relative to our expectations, but also to the rest of the group. The students have multiple opportunities to pass SOBs. There are no deadlines and SOBs can be demonstrated anytime during the year, although each SOB carries a "suggested" date range in which it should be observed. Although the formal aspect of the profiling method appears to be a tick-box exercise, discussion and written comments (where appropriate) are provided at several points throughout the year.

4.1 The Student Observable (SOB) Tool

Overall, we have defined 119 SOBs: 34 threshold, 50 typical and 35 excellent. In terms of Racket-specific SOBs, 10 of them are threshold and include behaviours such as "Use define, lambda and cond, with other language features as appropriate, to create and use a simple function."; 15 SOBs are typical, such as "Define functions to write the contents of a data structure to disk and read them back"; there are 13 SOBs at the excellent level, for instance: "The student can build an advanced navigation system for a robot in Racket that uses different data streams"

Our first year cohort consists of approximately 120 students. An appropriate tool is crucially needed to keep track of the progress of each student and to alert the teaching team as soon as problems arise (students not attending, students not being observed for SOBs, etc.). We have developed an on-line application that takes care of this aspect, in collaboration with research associates in our department.

Figure 4 presents a screenshot of the tool when entering or querying SOBs. The first column identifies the SOB by number; the second the level (threshold, typical, excellent); the third the topic (Racket, Fundamentals, Computer Systems, Project Skills); the fourth offers a description; the fifth and sixth column indicate respectively start and expected completion dates; the last column is an edit option. In addition to this facility, the tool provides a set of graphs to monitor overall progress and attendance. Background processes generate reports for the teaching team about non-attending or non-performing students. As an example, Figure 5 shows in tabular form the list of students (id number, first and last name, email), highlighting those who have (threshold) SOBs that should have been observed at the current date.

Figure 6 shows a screenshot of the "observation" part of the tool. In this case a demo student is selected and then the appropriate SOBs can be searched using the filters on the right. Different colours are used to highlight the most relevant SOBs. In addition, for each level a progress bar displays the overall progress of the student in green against the overall average progress of the cohort (vertical black bar);

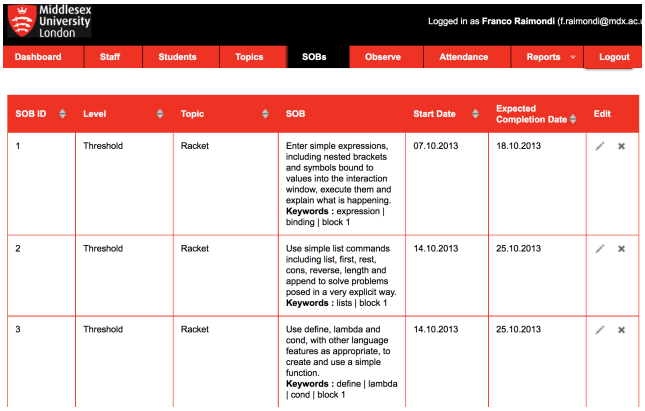

Figure 4: Entering and searching SOBs

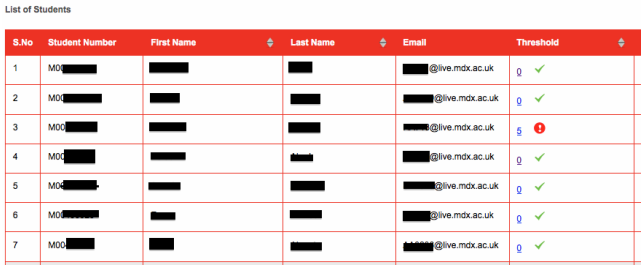

Figure 5: Student list with SOBs

in this case, the student is slightly ahead of the overall class for threshold SOBs. The "Notes" tab can be used to provide feedback and to record intermediate attempts at a SOB. In addition to the design presented in the figure we have also implemented a tablet-friendly design to be used in the labs.

Students are provided a separate access to the database to check their progress. A dashboard provides immediate and quick access to key information (number of SOBs expected to be observed in the coming week, number of SOBs that are "overdue", etc.). More detailed queries are possible for self-assessment with respect to the overall set of SOBs and with respect to the cohort in order to motivate students. As an example, Figure 7 shows the student progress (green bar) with respect to the whole class (yellow bars) for *typical* SOBs.

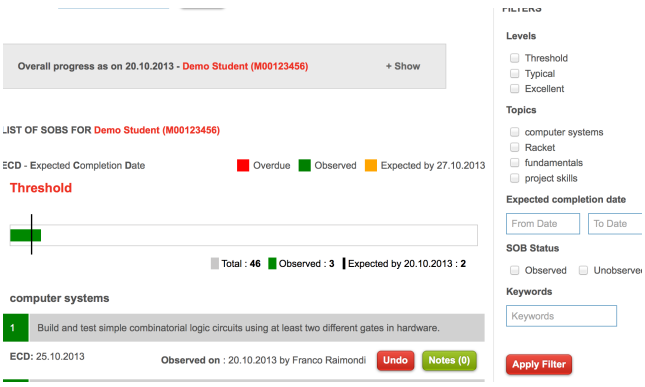

Figure 6: Observing a SOB for a student

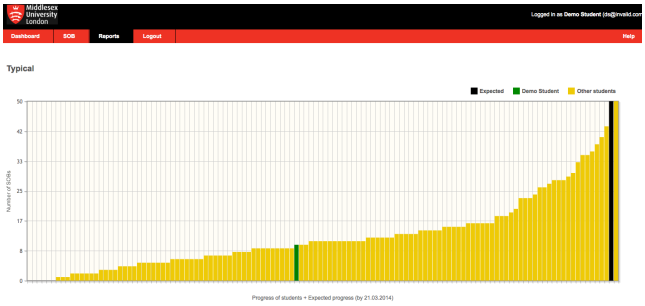

Figure 7: Student view: position with respect to class

As described in the following section, this tool has enabled the teaching team to provide continuous support to the students who needed it most, by identifying non-attending or dis-engaged students very early in the year.

5. EVALUATION

We provide here an overview of two forms of evaluation: a list of students' projects built using Racket and MIRTO, and an evaluation of average attendance, progression rate and engagement.

5.1 Student projects

In the final 3 weeks of their first year, students have been asked to work in teams and submit projects using MIRTO and Racket. Members of staff have provided support, but all the projects have been designed and implemented entirely by the students. The following is a list of some of these final projects.

- Dancing robots: this has been a popular theme, with more than one group working at coordinating the movement of multiple robots in a choreography of their choice. Two example videos are available at `https://www.youtube.com/watch?v=V-NfC4WK2Sg` and `https://www.youtube.com/watch?v=nMjdH9TCKOU`.
- A student has developed a GUI running on the Raspberry Pi. By tunnelling an X connection through SSH the robot can be controlled from a remote computer. The project also includes the possibility of taking pictures and a sequence of instructions to be executed. The video is available at the following link: `https://www.youtube.com/watch?v=FDi2TSCe3-4`
- A student has implemented a web server running on the Raspberry Pi, so that the robot can be controlled using a browser. The web interface enables keyboard control of the movements and detects the values of infra-red and bump sensors. Additionally, from the web interface a user could take a picture or start line following (on a separate thread). Finally, the student has also implemented a voice recognition feature by combining Racket and Pocketsphinx [5]: when the name of a UK city is pronounced, the local weather is retrieved. The video is available at this link: `https://www.youtube.com/watch?v=lwsG0lD55wk`.
- Finally, a student has taken a commercially available robotic platform (4tronix initio robot) built on top of

Arduino and has modified it by installing firmata and by adding a Raspberry Pi running Racket. To this end, the student has developed a bespoke version of MIRTOlib.rkt for this new robotic platform, adding support for servo motors. The video of this project is available at this link: `https://www.youtube.com/watch?v=hfByxWhyXkc`.

More importantly, through the projects and the threshold SOBs we have been able to assess the ability of nearly all students to control a robot from Racket, thus ensuring that they have achieved the minimal level of familiarity with the language to progress to the second year.

5.2 Attendance, engagement and progression

The teaching team has been concerned with various risks associated to this new structure of delivery for a whole first year cohort:

- Would students *attend* all the sessions, or only drop-in to tick SOBs?
- Would students *engage* with the new material?
- Would students focus on threshold SOBs only, and not *progress* beyond this level?

The delivery of this year has now nearly completed, with only two weeks left in our academic year. In "standard" programmes these are typically dedicated to revision before the exams. In our case, instead, we are in a position of analysing the data collected over the year to answer the questions above.

5.2.1 Attendance

Figure 8 shows the weekly attendance rate in percentage for the new first year programme (in blue) and for two other first year modules from another programme (in green and red, anonymised). Unfortunately, no aggregated attendance data is available for the other programme. As a result, we can only compare attendance of the whole first year with these two modules, one of which has *compulsory* attendance.

The graph displays attendance per week; a student is considered to have attended in a week if s/he has attended *at least one session* during the week. "Standard" modules have an attendance ranging between 50% and 70% for the "core" module with compulsory attendance, and between 40% and 60% for the "non-core" module. There is also a decreasing trend as weeks progress.

We have been positively surprised by the attendance for the new programme, which has been oscillating between 80% and 90% with only a minimal drop over the year (the two "low" peaks around week 10 and 17 correspond to British "half-term" periods, when family may go on holiday). Unfortunately, no aggregated attendance data is available for other programmes. As a result, we can only compare attendance of the whole first year with a *compulsory* module in another programme and for a standard first year module.

5.2.2 Engagement

Engagement is strictly correlated with attendance, but it may be difficult to provide a direct metric for it. We typically assess engagement by checking log-in rates in our VLE environment and, in our case, we could also measure

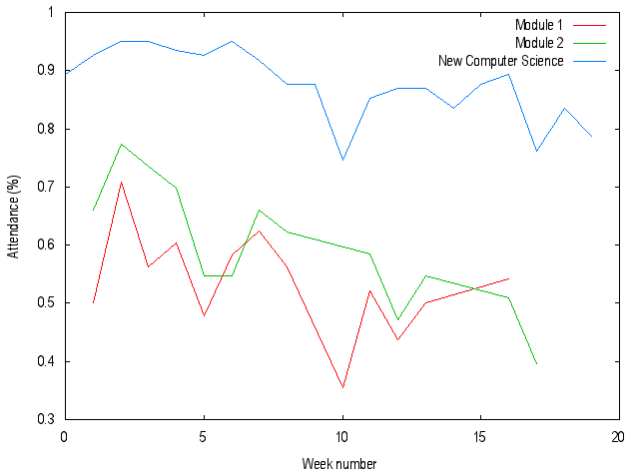

Figure 8: Weekly attendance (comparison)

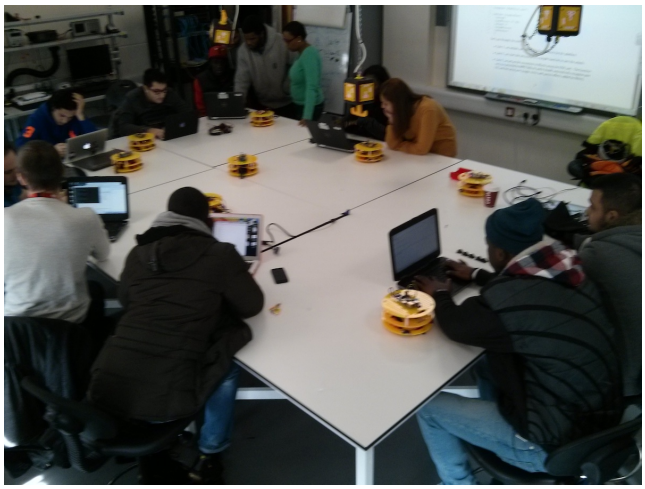

Figure 9: Example lab session

SOB progression. We were able to identify approximately 10% of the cohort being "not engaged". Thanks to our tool, we have been able to address these students individually.

In addition to SOB progression, we could also measure usage of the MIRTO platforms. We have built 10 yellow and 10 blue robots. We have used 4 of these for research and 2 for demo purposes, leaving a total of 7 blue and 7 yellow robots for teaching in the workshops. There are typically 20 students allocated to each workshop, working in groups of 2 or 3 (see Figure 9); all sessions required all robots, showing that all students were engaged with the material.

5.2.3 Progression

Finally, there was a risk that the majority of the class would focus just on the achievement of threshold SOBs. Our first year is not graded and therefore, once the threshold SOBs have been achieved, there is no formal difference between students with different numbers of SOBs.

Besides anecdotal evidence of students working on optional projects, our monitoring tool has allowed us to encourage the best students to work on new challenges for the whole year. This has resulted in the vast majority of stu-

dents progressing beyond the "threshold" level. This is confirmed by the results presented in Figure 10: the majority of students has progressed well beyond the 34 threshold SOB mark (red line in the figure). The same trend is confirmed if Racket-specific SOBs are considered. Figure 11 shows that approximately 70% of the students have completed SOBs beyond the required threshold level (the same distribution occurs for other SOB categories).

The tool has also shown interesting approaches to this new structure, both in general and for Racket-specific SOBs: some students have focussed on threshold SOBs first and only moved to typical and excellent SOBs later. Other students, instead, have worked at typical and excellent SOBs with many threshold SOBs still outstanding.

6. CONCLUSION

In designing a new Computer Science programme for Middlesex University we have decided to make use of Racket and to design and build a robotic platform to support our delivery. To the best of our knowledge, this is the first time that this approach is applied at such a large scale. The preparation of this new programme has required the joint effort of a large team of academics and teaching assistants for more than a year before the actual delivery. However, the results obtained are very encouraging: attendance and engagement are well above average, and the large majority of students are progressing beyond the level required to pass this first year.

7. REFERENCES

[1] Harold Abelson and Gerald J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, USA, 2nd edition, 1996.

[2] Massimo Banzi. *Getting Started with Arduino*. Make Books - Imprint of: O'Reilly Media, Sebastopol, CA, ill edition, 2008.

[3] The Firmata protocol. `http://firmata.org/`. Accessed: 2014-03-20.

[4] The MIddlesex Robotic plaTfOrm (MIRTO). `http://www.creative-robotics.com/About-HUBee-Wheels`. Accessed: 2014-03-20.

[5] David Huggins-daines, Mohit Kumar, Arthur Chan, Alan W Black, Mosur Ravishankar, and Alex I. Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *in Proceedings of ICASSP*, 2006.

[6] Myke King. *Process Control: A Practical Approach*. John Wiley & Sons, 2010.

[7] M. Loomes, A. Jones, and B. Show. An education programme for software engineers. In *Proceedings of the First British Software Engineering Conference*, 1986.

[8] Martin Loomes, Bruce Christianson, and Neil Davey. Formal systems, not methods. In *Teaching Formal Methods*, pages 47–64. 2004.

[9] M. Margolis. *Arduino Cookbook*. O'Reilly Media, 2011.

[10] The MIddlesex Robotic plaTfOrm (MIRTO). `https://github.com/fraimondi/myrtle`. Accessed: 2014-03-20.

[11] The Racket Language. `http://racket-lang.org`. Accessed: 2013-10-21.

[12] Racket Firmata for Middlesex Students. `https://bitbucket.org/fraimondi/racket-firmata`. Accessed: 2014-03-20.

[13] Racket Firmata. `http://planet.racket-lang.org/display.ss?package=firmata.plt&owner=xtofs`. Accessed: 2014-03-20.

[14] Fan Yang, Frederick W. B. Li, and Rynson W. H. Lau. A fine-grained outcome-based learning path model. *IEEE T. Systems, Man, and Cybernetics: Systems*, 44(2):235–245, 2014.

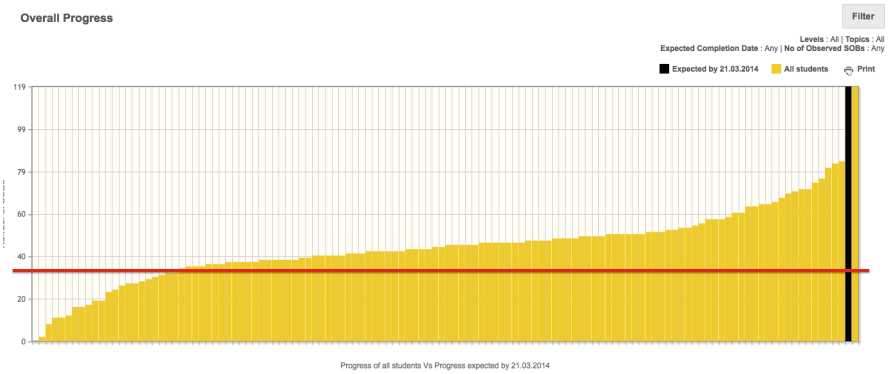

Figure 10: SOB overview (end of year)

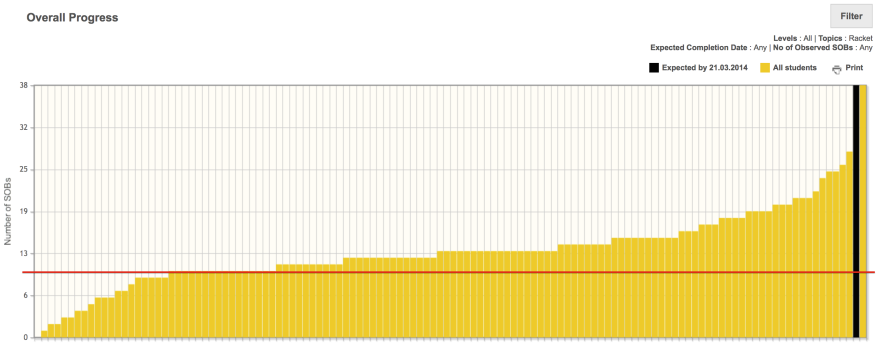

Figure 11: Threshold SOBs for Racket (end of year)