Flipped Graduate Classroom in a Haskell-based Software Testing Course

Jan van Eijck
Software Analysis and Transformation
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
jve@cwi.nl

Vadim Zaytsev
Informatics Institute
Universiteit van Amsterdam
Amsterdam, The Netherlands
vadim@grammarware.net

Extended Abstract

Master Software Engineering[1] is an intensive one year programme of graduate education at the University of Amsterdam. Admission is based on having an Bachelor degree or its equivalent, as well as on demonstrating advanced levels in skills like software development, logic, term rewriting, compiler construction and algorithmics. *Software Specification and Testing* is one of the six core courses of the programme. At the entry level, students are expected to be able to manipulate simple mathematical formulae and have some basic knowledge of (semi-)automated testing techniques. The learning objectives of the course, in Bloom's (revised) terms [1], are:

- to recognise various testing techniques applied in practical software engineering;
- to compare testing techniques by applicability in certain scenarios;
- to implement formal specifications of software systems and test such systems for conformance;
- to differentiate among alternative approaches to testing and argue in favour of one against another;
- to judge efficiency of a given model for testing.

In previous years the course has been refined and improved with each run (according to both self-assessment and student evaluation), and in 2013 its components for two months were:

- a weekly lecture, in the classic sage-on-the-stage model, with one group of 60 students;
- an irregular complementary material from the book [4];
- a weekly workshop of two hours, in groups of 20 students, for questions and answers, as well as for practicing the skills from the mathematical side of the material;
- a day and a half long programming practical studies, with weekly sets of assignments to be solved in Haskell [9] in groups of 2–5 students.

The final grade in the course was determined by a final theoretical written examination (30%), a final practical examination in Haskell programming (30%) and a cumulative score of the assignments completed by a group during the semester (40%). The course was positively evaluated by the students (an average score for each question at least at 3 out of 5), yet with a disturbing number of remarks stating that they still do not see how specifications relate to the practice of software engineering.

[1] http://www.software-engineering-amsterdam.nl

Submitted to:
TFPIE 2014

The *Academic Skills* course ran in parallel to *Software Specification and Testing* as preparation for the final Master's project. The students learnt the basics of critical thinking and acquired skills on literature seeking, filtering, assessment and synthesis. The courses were not strongly linked, but the assignments for the *Academic Skills* involved research papers related to testing techniques.

In retrospective, the following aspects of the course were already close in spirit to flipped classroom [10, 11] or felt like they could be improved by going for a systematic flip:

- many students, especially those initially weaker in functional programming, took complementary reading material very seriously and in the end regretted that it was not always the case that a lecture was linked to particular chapters of the book;
- most time spent in classroom was dedicated, both by design and de facto, to solving practical assignments and acquiring programming and specification skills, not on studying the underlying theory;
- two most appreciated components of the course by the students were workshops where they got direct feedback and were solving assignments in direct and close collaboration with the teachers, and detailed feedback on their Haskell code;
- we put some relatively advanced technology to use: each student group had its own git repository for version control of their source code — the teachers had access to all repositories and were able to leave feedback remotely by commenting on commits, making pull requests, reporting issues, etc;
- some kind of ad hoc learning analytics was used to resolve inter-group problems, and individual grades within a group in several cases needed to be adjusted based on commits made by each person;
- most problems we experienced were related either to heterogeneous backgrounds of students (ranging from complete novices in functional programming to having had strong Haskell courses in the past) or to inadequacy in meeting their expectations by insufficiently demonstrating the links existing between the theoretical part (mathematical induction, equivalence classes, lambda calculus, type systems) and the everyday practice of a future software engineering practitioner.

Thus, the next edition of the *Software Specification and Testing* course would involve the following:

- a stict requirement to complete some functional programming course before being admitted to the programme — those without such experience would be asked to complete a self study premaster course by reading one of the well-known entry-level Haskell books [4, 5, 6, 8, 7, 12, 13, 14, 15];
- every lecture would be disassembled into several related topics that would be required for students to study on their own (we would provide video recordings and other material);
- each topic would have several examples attached, with the objective of allowing students to choose the most understandable example or a combination of examples (their usage would be monitored to be analysed later);
- the preparation phase should be possible to skip entirely for extremely knowledgeable students or to skip partly for students familiar with some of the topics, yet with all necessary information provided to learn the material from scratch, given enough time and effort;
- each topic would have a (possibly automated) set of questions one is expected to be capable of answering, for self-assessment purposes and for measuring progress;

- each topic would be accompanied by at least one open question without a clearly defined answer, to encourage the students to think deeper about the topic and to serve as a reference point for other possible questions to be discussed in class;
- instead of a lecture and a workshop there would be one session dedicated to open discussion and questions and answers, and one session for solving exercises and emphasizing bidirectional mapping between the math apparatus and executable Haskell code; both would be accompanied by notes, remarks or summaries posted after the session;
- several times per course there would be guest lectures by functional programming practitioners (mostly Haskell and Erlang) from companies applying the techniques explained within the course;
- introducing the actual process of programming in Haskell would be done by self-studied examples plus a hack sprint, a mini-hackathon of 2–4 hours with advanced assignments solvable within allocated time, with several valid solutions; we expect collaborative discussion, coding in pairs and brief presentations of the results;
- weekly assignments would have a clearly defined competition target, possibly changed every week: bonus points would be given to the author of the fastest solution, or the most concise one, or the one with the least side effects, or one making best use of a fixed point combinator, etc; having this defined upfront would help clarify the expectations up to the point of guaranteed self-assessment;
- (part of) the *Academic Skills* course would be integrated with *Software Specification and Testing*: students would be asked to pick a testing approach from a list, investigate it, write a short review and present it; the reviews would be peer assessed, since there seems to be strong positive evidence for groups of similar size and level [16]; the presentations are supposed to give all the students a systematic overview of the field;

In general, by flipping the classroom in one of the most difficult (reportedly) courses in a Master's programme, we hope to make better utilisation of teachers by spending time in class discussing the questions actually raised after studying the material; we hope to provide means to students to pace their own learning process and thus ensure some minimum learnt by everyone disregarding their initial level; and we certainly hope to collect more data from diagnostics and analytics in order to keep improving the course in future years. Given that the graduate students usually do not suffer from the lack of motivation (which is commonly presented as one of the biggest downsides of the flip [2, 3]) and that all elements of the flipped education already present, were met with enthusiasm, we have high expectations for this course.

References

[1] Lorin W. Anderson, David R. Krathwohl, Peter W. Airasian, Kathleen A. Cruikshank, Richard E. Mayer, Paul R. Pintrich, James Raths & Merlin C. Wittrock (2000): *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, 2 edition. Allyn & Bacon.

[2] Katie Ash (2012): *Educators View "Flipped" Model With a More Critical Eye*. Education Week 32(2).

[3] Jonathan Bergmann & Aaron Sams (2012): *Before You Flip, Consider This*. Phi Delta Kappan 94(2).

[4] Kees Doets & Jan van Eijck (2004): *The Haskell Road to Logic, Maths and Programming*. Texts in Computing 4, King's College Publications.

[5] Jan van Eijck & Christina Unger (2010): *Computational Semantics with Functional Programming*. Cambridge University Press.

[6] Eric Etheridge (2011): *Haskell Tutorial for C Programmers*. OpenLibra. Version 3.0. `http://www.haskell.org/haskellwiki/Haskell_Tutorial_for_C_Programmers`.

[7] Paul Hudak (2000): *The Haskell School of Expression — Learning Functional Programming through Multimedia*. Cambridge University Press, New York. `http://www.cs.yale.edu/homes/hudak/SOE/`.

[8] Graham Hutton (2007): *Programming in Haskell*. Cambridge University Press.

[9] Simon Peyton Jones, editor (2003): *Haskell 98 Language and Libraries. The Revised Report*. Cambridge University Press.

[10] Alison King (1993): *From Sage on the Stage to Guide on the Side*. College Teaching 41(1), pp. 30–35, doi:10.1080/87567555.1993.9926781.

[11] Maureen J. Lage, Glenn J. Platt & Michael Treglia (2000): *Inverting the Classroom: A Gateway to Creating an Inclusive Learning Environment*. The Journal of Economic Education 31(1), pp. pp. 30–43.

[12] Miran Lipovaca (2011): *Learn You a Haskell for Great Good! A Beginner's Guide*. No Starch Press. `http://learnyouahaskell.com`.

[13] Bryan O'Sullivan, John Goerzen & Don Stewart (2008): *Real World Haskell*. O'Reilly Media, Inc. `http://book.realworldhaskell.org`.

[14] Michael Snoyman (2012): *Developing Web Applications with Haskell and Yesod*. O'Reilly Media, Inc.

[15] Simon Thompson (1999): *Haskell: The Craft of Functional Programming*. Pearson Education. `http://www.haskellcraft.com/craft3e/Home.html`.

[16] Andrii Vozniuk, Adrian Holzer & Denis Gillet (2014): *Peer Assessment Based on Ratings in a Social Media Course*. In Matthew D. Pistilli, James Willis, Drew Koch, Kimberly E. Arnold, Stephanie D. Teasley & Abelardo Pardo, editors: Learning Analytics and Knowledge Conference (LAK '14), ACM, pp. 133–137, doi:10.1145/2567574.2567608.