10 Years of Agile Lab Courses for International Students

Daniel Speicher, Pascal Bihler, Paul Imhoff, Günter Kniesel, Holger Mügge,
Jan Nonnen, Tobias Rho, Mark von Zeschau, Armin B. Cremers

Computer Science III, University of Bonn, Römerstraße 164, 53117 Bonn
{dsp, bihler, gk, muegge, nonnen, rho, mvz, abc}@cs.uni-bonn.de

Abstract: The Institute of Computer Science III of the University Bonn regularly offers Agile Lab Courses to students from Germany and all over the world as part of the International Program of Excellence at the Bonn-Aachen International Center of Information Technology. In the recent ten years we offered about 16 courses with a duration of four to six weeks. Typically around twelve students are introduced into Agile Software Development by one and a half to three colleagues. During this time the team of students develops software of realistic complexity that is of real value for a research project or an external customer.

1 Experiencing the Real Thing

In the article that is commonly seen as one of the founding documents of the waterfall approach to software development [Roy70] Winston W. Royce wrote: “The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable.” Although he recommends thorough analysis and careful design he was very aware of the dangers of late insight into unforeseen problems. In a way most of our students are in the same situation with respect to their software technology knowledge. They get an overview of traditional and Agile approaches and are able to reproduce their knowledge to pass an exam. Yet, they do not have the experience whether their understanding is strong enough to contribute to a real software product. This is what our courses [1] are out to offer them.

2 Early Substantial Feedback

Besides technical challenges, long development cycles lead to the problem of outdated business needs and customer expectations. This is why Extreme Programming and Scrum focus so much on the value of *early feedback*. In one of the first days we ask the students to implement some functionality that cuts through all technological layers so that they get some first feedback about the required technologies. The functionality of this *vertical slice* naturally needs to be extremely simple. The usage of *tests* is strongly encouraged and the students love the traffic light that makes the result of the recent run of the *continuous integration* server so visible.[2] We make sure that the students understand that software development is not only about solving some

[1] `http://sewiki.iai.uni-bonn.de/teaching/labs/archive`

[2] Introducing continuous integration was one of many improvement suggestions that we got over the years from an external industry consultant (`http://www.michaelmahlberg.de/`).

tricky technical tasks but about *providing value* to someone who would - outside the academical setting - honor the production of value by paying their salaries. To represent this perspective one colleague acts as a *customer* [MSK04] evaluating the value of different functionalities while leaving the technical decisions to the students (and the other colleague(s)). We introduce this perspective using the *XPGame*[3] and encourage to present the results of an iteration truthfully from the perspective of the customer.

3 Reflective Improvement for Responsible Students

Software of realistic complexity requires a broad range of knowledge. We leverage on the number of students by asking everyone to *become an expert* in one area. The seminar, a *wiki as a knowledge base* and consequent *pair programming* allows the knowledge to expand into the team. The students are offered the responsibility to *estimate* the effort a certain functionality requires. With our support they learn that they can accept this responsibility. *Planning Poker* during planning is very helpful in this respect and some progress *tracking* gives feedback about the quality of their estimates. Daily *stand-up meetings* make sure that problems become obvious early and everyone shares the latest insights that are of value to the team. Each iteration ends after the presentation to the customer with a *retrospective*. In a respectful atmosphere everyone can the discuss the challenges freely and actions addressing them can be found.

4 What did we learn?

The students are typically very satisfied with the course, praise the friendly and productive atmosphere and are surprised by the improvement of their skills. Nevertheless we are not always successful from the start, so that we can offer as well some lessons learned: There need to be some breadth in the product so that two or more somewhat independent functional areas can be addressed by the students in parallel. Setting up a testing framework for a complex technology is beneficial and can be supported but not completed by a single course of four weeks. There is a limit to what can be estimated appropriately. If the team is in doubt that something takes more than two days it is probably too big to be estimated. Advocating that finishing functionality should be favored over investing into new one is not enough as the final integration seems to tedious and challenging. Visualizing the workflow and placing limits on the work in progress finally solved this issue for us [NIS13].

Literatur

[MSK04] H. Mügge, D Speicher und G. Kniesel. Extreme Programming in der Informatik-Lehre - Ein Erfahrungsbericht. In *Informatik 2004 - Beiträge der 34. Jahrestagung der GI, Lecture Notes in Informatics*, 2004.

[NIS13] J. Nonnen, J. P. Imhoff und D. Speicher. Kanban im Universitätspraktikum - Ein Erfahrungsbericht. In Andreas Spillner und Horst Lichter, Hrsg., *SEUH*, Jgg. 956 of *CEUR Workshop Proceedings*, Seiten 91–98, 2013.

[Roy70] W.W. Royce. Managing the development of large software systems: concepts and techniques. *Proc. IEEE WESTCON, Los Angeles*, Seiten 1–9, August 1970. Reprinted in *Proceedings* of the Ninth International Conference on Software Engineering, March 1987, pp. 328–338.

[3] `http://www.xp.be/xpgame.html`