8th International Conference on Higher Education Advances (HEAd'22)
Universitat Politècnica de València, València, 2022
DOI: http://dx.doi.org/10.4995/HEAd22.2022.14672

Dealing with Uncertainty – Experiencing Real Life in Class

Daniela Zehetmeier[1], Axel Böttcher[2]
[1]Lufthansa Aviation Training GmbH, Südallee 15, D-85356 München-Flughafen, Germany, [2]HM Munich University of Applied Sciences, Department of Computer Science and Mathematics, Lothstraße 64, D-80335 München, Germany.

Abstract

It is likely that our Computer Science graduates will be confronted with software which has been growing over a long period of time. In order to master resulting challenges in their later professional lives, students need to be able to deal with the inherent uncertainty of legacy software systems. Observations show that many students are bad at dealing with uncertainty. Therefore, it is important to address the competence of dealing with uncertainty in teaching.

In this article, we describe our experiences with addressing this important competence in teaching of a module on Software Archaeology.

The basis is to establish a teaching and learning environment that creates uncertainty within lab sessions. We achieved this by using a project from industrial practice. This, however, also induces uncertainties regarding the teaching and assessment processes.

We conclude that further methods need to be developed to address this competence with respect to teaching and assessment.

Keywords: *Higher cognitive abilities; uncertainty, software maintenance and evolution; sustainable software.*

1. Motivation

The training of our students in software development and software engineering requires dealing with a large number of abstract concepts and formal language constructs that can be combined with each other in almost any way. We have observed that we tend to focus strongly on the competence levels of understanding and application of constructs. Although we usually formulate learning objectives for our courses also at higher competence levels according to Bloom's revised taxonomy of learning objectives (Anderson et al., 2001), important learnings related to higher levels of competency are often neglected.

An example of such a neglected skill is the ability to deal with uncertainty. Many of our teaching examples and practical tasks in the first semesters are based on crystal-clear requirements, so as not to allow uncertainties to arise. This results from large freshmen cohorts where feedback has to be produced in an efficient manner. We also try not to make ourselves vulnerable in the evaluation by keeping the space for discussions as small as possible. This leads to a stronger schoolification which is not the goal of higher education. Also in practice, software developers are often faced with unclear requirements and old undocumented legacy software. Such environments bring up uncertainties, they have to deal with.

2. Objectives

Based on these findings, we want to adapt our teaching and address the deficits mentioned in advanced courses. The above points can be integrated into teaching by providing the students with an extensive existing codebase. The less such a project is in compliance with the classical quality standards, the more we have to work like archaeologists would put it: “To draw knowledge from what has been created by humans in ancient times” (Sinn, 2000). Starting from such a metaphorical allusion, Hunt and Thomas raised the term software archaeology already in 2002 (Hunt & Thomas, 2002).

In this article, we use the example of a module on software archaeology to describe our experience with these important topics in the field of software engineering.

3. Literature

Software maintenance is definitely mentioned in standard curricula. It is often considered as a sub-discipline of software engineering but rarely seen as a separate module; see e. g. (ACM, 2016).

In 2003, van Deursen et al. state in a workshop contribution for Program Comprehension: “Students learn how to write new programs but they are not taught how to read and change existing and large ones” (Deursen et al., 2003).

Both Smith and Pinto follow the approach to teach maintenance and evolution by using existing open source projects (Smith et al., 2014, Pinto, 2017). We extend this field to software archaeology by a retrospective architecture and requirements analysis.

Overoye and Storm (Overoye & Storm, 2015) collected evidence that students can benefit from experiencing uncertainty and from having the opportunity to overcome it. They claim, however, that "it is the process underlying the change from uncertainty to certainty that leads to deeper understanding and better memory for the to-be-learned information".

4. Course Design

The module on Software Archaeology is designed as an elective subject with five ECTS for the bachelor's degree programs in Computer Science as well as Information Systems and Management. It is designed for two hours per week each: lecturing and lab time. Assessment is done in two-parts: a graded study paper and a graded oral exam.

Due to Covid-19, the course had to take place completely online in the summer semester 2021. We had the opportunity to teach the module in pair-teaching mode (Zehetmeier et al., 2018).

4.1. Learning objectives

We formulated the following learning objectives based on the competencies that are required for the given tasks:

- You analyze existing code to understand it and
  - to draw conclusions about the intention of the original developers
  - to identify requirements so that they can be used as a basis for refactorings or a re-implementation
- You document the knowledge gained using suitable tools.
- You apply reverse engineering techniques systematically and purposefully.
- You apply refactoring techniques systematically and purposefully.
- You analyze control flow theoretically and based on existing initial data.
- You will design, implement and execute tests for legacy code.
- You discuss procedures and results in your lab group and in plenary.

4.2. Project used within the module

The focus of the course should be an existing project that serves as a guideline for a substantial discussion within a semester. According to Smith et al. (Smith et al., 2014) we formulated criteria for such a project in advance:

- The project must be sufficiently large and complex so that no group of students tries to carry out a complete re-implementation on a single weekend.
- The project must have technical debts.
- The project must deal with various external systems and interfaces.
- The project should not come from an academic context.

In this respect, projects from the open source environment could be considered, as well as projects from practice. Unlike described in (Pinto et al., 2017) we just didn't want an Open Source community, that could be contacted as a fallback in case of difficulties. The change of one of the authors to industry opened up the possibility of working on several projects from the professional context that meet the requirements. We chose a historically grown, approximately twelve-year-old project realized in ColdFusion. The software is still in productive use. The scope is about 30,000 lines of code. The software deals with several external systems and interfaces, is largely undocumented, has no tests, and brings various more technical debts with it.

4.3. Topics and Tasks

The existing software should be analyzed in the first two thirds of the semester with the goal of creating an architecture documentation based on the arc42 template (Starke et al., 2019). This includes a description of the requirements that can be extracted ex-post from the software. In addition to a support for maintenance work, the documentation can provide a basis for the upcoming reimplementation of the system. Finally, a modern user interface should be designed so that the benefits of archaeological work become visible.

The module is comprised of the following thematic blocks:

Glossary In addition to setting up the project, the first block included creation of a glossary. The task was to continuously expand the glossary with knowledge gained over time.

Extraction of an API documentation This task required the creation of a documentation of the project's (pre-REST) HTTP-API including a description of the chosen approach to this task in a wiki. The concrete design of this documentation was left to the students.

Interface description and cross-cutting concerns Creating the arc42 template and to fill it with glossary and API documentation was the third task. Additionally, stakeholders, boundary conditions and context had to be identified, as well as any cross-cutting concerns and external interfaces of the system "as far as possible".

Documentation of the database In this step, the relationships between the existing database tables had to be analyzed and documented by reverse engineering.

Runtime view After having thoroughly dealt with the static view of the application, the students should determine dynamic views. Individual API backend functions had to be described on an adequate high level of abstraction.

Documentation of requirements An ex-post extraction of the requirements was considered as the basis for a later re-implementation of the application.

GUI At the end of the semester, students should propose a redesign based on the knowledge acquired during the semester.

5. Observations and Reflection

When designing the course, it was important to us that the project contains many uncertainties. The historically grown software also confronted us lecturers with a considerable amount of uncertainty. The complete range of functionality and structure had not fully opened up to us either.

During the preparation of the individual lectures as well as in the retrospective, we repeatedly discussed various statements, work results, and also the behavior of students. Two findings appear worth a special discussion at this point: assignments that appear vague, simply because they do not state volume of work required, and how to deal with the resulting uncertainty.

5.1. Dealing with tasks that are vague with respect to quantity

The tasks were vague with respect to volume as even we lecturers had to build hypotheses and verify them in a critical discussion.

Example: ”Describe external interfaces as far as possible”. Already the lack of a quantitative statement created a feeling of uncertainty. Therefore, the students repeatedly demanded quantitative statements, such as how much they have to do to pass the module. We couldn't determine the exact number of external interfaces ourselves. Thus, we explained that we put emphasis on the students’ solution approach in the final grading.

During the semester we kept asking ourselves the question whether we needed to know the project better, in order to make such quantitative statements. However, we repeatedly came to the conclusion that it is precisely the lack of knowledge that brings the project close to a real situation enabling all possibilities of learning by uncertainty (Overoye & Storm, 2015). The evaluation at the end of the semester revealed a very heterogeneous picture among the students with regard to their ability to deal with the occurring uncertainties:

"The course is very realistic, which in my opinion increases the relevance of its content. That's exactly why I personally think the course is very interesting."

"The tasks were not posed well and there was a lot of discussion about what exactly had to be done."

How do we create sufficient certainty for the students despite the naturally vague work assignments, so that they are not in a constant state of vagueness with regard to their grades?

5.2. Impact of uncertainty on quality of results

A comparison of a task's level of uncertainty and the quality of results confirms our hypothesis that the quality of results decreases with increasing uncertainty:

Task Glossary: high degree of uncertainty, poor overall rating. At the beginning of the semester, the assignment to create a glossary offered a high degree of certainty to the students: Some terms from the context of the project have been addressed in the course. But we rapidly reduced the number of explicit hints towards which terms to add to the glossary. Students underestimated the importance of terms from the technical context of the application and neglected expanding the glossary. Overall, the resulting glossaries were of inferior quality. Students did not sufficiently differentiate which terms are important and which are not. Many glossaries were merely lists of acronyms.

Task API documentation: low level of uncertainty, good overall rating. Students did not face a high amount of uncertainty during the creation of the API documentation. The task could also be solved through diligence. The students' results were consistently rated very well.

Task Interfaces: medium degree of uncertainty, medium overall rating. One task towards the arc42 documentation was the identification and description of the external interfaces. Here we did not make a quantitative statement on how many interfaces exist and thus need to be identified. The students faced a degree of uncertainty since they had to decide for themselves when to finish their research. The rating of this task is mediocre. We suspect that the positive trend results from the good search mechanisms and the use of standardized interfaces (e.g. HTTP requests).

Requirements: high degree of uncertainty, poor overall rating. Students had major problems with a description of the requirements for the software functions they had to analyze. To imagine which requirements are the basis for the functions and to describe them in the context of the system was difficult for all groups. The number of questions were also at a peak during this exercise. Despite numerous discussions, the task could only be completed with a rather poor overall result.

The evaluation results show the existing uncertainty among students:

”Project from reality, even if it's not nice to work with ColdFusion”

”[...] the project with Lufthansa was somehow only suitable to a limited extent, since many challenges/tasks could only be solved by guesswork”

“The confusing code and bad coding style make analyzing the project laborious. This may be an accurate representation of reality, but it is motivating not to work out the lab assignments beyond the minimum.”

There was a very heterogeneous mix of students, independent of their semester. Some can deal with the uncertainty of the task – others can not. As a result, we lecturers have to specifically teach how to deal with uncertainties.

If you teach how to deal with uncertainties, you should assess these competences according to the principle of constructive alignment (Biggs, 1996). But how do we assess the competence of dealing with uncertainties? What are the criteria for objectively measuring the competence and how to communicate the assessment criteria transparently?

6. Discussion and Outlook

In this article we presented our observations on dealing with uncertainties in class on Software Archaeology. An important experience is that a grown extensive external project that was developed without clear quality standards, offers good conditions for this approach. The company cooperation has proven to be helpful in this context: the industrial project does not come from the professor’s ”weird world of thoughts”, which makes the students perceive legitimacy, credibility and authenticity. With this project we were able to provide a task with a large space of possible solutions.

Nevertheless, our students tend to push for clear answers or process descriptions that they can internalize for the exam or use in their study paper. If we resist to provide this, they feel great uncertainty. Here a dilemma arises for the lecturers: if they give too much and too detailed feedback too early, students adapt work results to a solution, lecturers have in mind – a result that is to be avoided. From the student's point of view the lack of early feedback takes away their opportunity to improve grades during the semester, even though we were willing to accept any reasonable solution with well argumented derivation.

Encouragement to continue on the approach chosen, appreciation for work results, and discussing the pros and cons of approaches and artifacts, help dealing with uncertainty in the project. On the other hand, artificially creating certainty does not foster the ability to last uncertainties now and in future professional life.

Supporting students in such a course where higher cognitive skills are addressed exceeds the usual time budget for a course significantly. We are missing efficient forms of supporting our students. This should be the next step in research about this competence.

In summary, several questions arise from this article: Which other methods are suitable for integrating uncertainty into teaching? And how can we objectively assess these skills so that the assessment criteria can be communicated? This could be a factor to lower the uncertainty regarding the exam performance. The students could then focus on the uncertainty the project context brings with it.

References

ACM (2016). Computer Engineering Curricula 2016 – Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. Tech. Rep., Assoc. for Comp. Machinery (ACM)/ IEEE Comp. Society

Anderson, Lorin W.; Krathwohl, David R.; Airasian, Peter W.; Cruikshank, Kathleen A.; Mayer, Richard E.; Pintrich, Paul R.; Raths, James; Wittrock, Merlin C., Hrsg. (2001). A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives. Longman, New York, 1. ed.

Biggs, Jhn (1996). Enhancing teaching through constructive alignment. Higher education, 32(3):347–364.

van Deursen, A.; Favre, J.-M.; Koschke, R.; Rilling, J. (2003). Experiences in teaching software evolution and program comprehension. In: 11th IEEE International Workshop on Program Comprehension, 2003. S. 283–284.

Hunt, Andy; Thomas, Dave (2002). Software Archaeology. IEEE Software, 19(2): 20–22.

Overoye, A. L., Storm, B. C. (2015). Harnessing the Power of Uncertainty to enhance Learning. Translational Issues in Psychological Sciences, 1(2), 140-148. doi: 10.1037/tp20000022.

Pinto, Gustavo Henrique Lima; Filho, Fernando Figueira; Steinmacher, Igor; Gerosa, Marco Aurelio (2017). Training Software Engineers Using Open-Source Software: The Professors' Perspective. In: 2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T). S. 117–121.

Starke, Gernot; Simons, Michael; Zorner, Stefan; Müller, Ralf D. (2019). arc42 by Example: Software architecture documentation in practice. Packt Publishing, Birmingham.

Sinn, Ulrich (2000). Einführung in die klassische Archäologie. C. H. Beck.

Smith, Thérèse Mary; McCartney, Robert; Gokhale, Swapna S.; Kaczmarczyk, Lisa C. (2014). Selecting Open Source Software Projects to Teach Software Engineering. In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education. SIGCSE '14, Association for Computing Machinery, New York, NY, USA, S. 397–402.

Zehetmeier, Daniela; Böttcher, Axel; Brüggemann-Klein, Anne. (2018). Designing Lectures as a Team and Teaching in Pairs. in Proc. 4th International Conference on Higher Education Advances (HEAd'18), Valencia. 873—880