Experiences in Teaching Program Transformation for Software Reengineering

Mohammad El-Ramly
mer14@le.ac.uk
Department of Computer Science,
University of Leicester, UK

Abstract

Little attention is given to teaching the theory and practice of software evolution and change in software engineering curricula. Program transformation is no exception. This paper presents the author's experience in teaching program transformation as a unit in a postgraduate module on software systems reengineering. It describes the teaching context of this unit and two different offerings of it, one using Turing eXtender Language (TXL) and the other using Legacy Computer Aided Reengineering Environment (Legacy-CARE or L-CARE) from ATX Software. From this experience, it was found that selecting the suitable material (that balances theory and practice) and the right tool(s) for the level of students and depth of coverage required is a non-trivial task. It was also found that teaching using toy exercises and assignments does not convey well the practical aspects of the subject. While, teaching with real, even small size, exercises and assignments, is almost non-feasible. Finding the right balance is very important but not easy. It was also found that students understanding and appreciation of the topic of program transformation increases when they are presented with real industrial case studies.

Introduction

Software development is rarely a "green fields" activity. It is likely the case that programmers, even if they are doing fresh software development, have to live with some legacy system(s) from the past that they have to understand, admire, take care of and evolve. This would require them to have knowledge and skills in the areas of program comprehension, evolution, maintenance, reverse engineering and reengineering, suitable to their work context. These areas have received a lot of attention from the research community, which resulted in an increasing number of projects, conferences and workshops on these topics. Unfortunately, software engineering curricula are significantly lagging behind in providing the necessary training on these topics. Most of the time students are trained on developing small size programs from scratch. They learn how to write new programs but they are not taught how to read and change existing and large ones [1]. Software engineering textbooks cover the topic of software change and evolution minimally, as a side topic (Compare, for example, the 6th and 7th editions of Software Engineering by Ian Sommerville. Chapters 26, 27 and 28 in the 6th edition [9], which are titled Legacy Systems, Software Change and Software Re-engineering, respectively, were reduced to one chapter in the 7th edition, Chapter 21: Software Evolution [10]).

There is a need for more emphasize in software engineering educational programs on the issue of software evolution and change. There is also an equal need for coherent packages of educational materials of different levels of depth to serve different curriculum contexts.

In this paper, I share my experience in teaching one aspect of software reengineering, which is the application of program transformation in reengineering, to M.Sc. of software engineering

Dagstuhl Seminar Proceedings 05161
Transformation Techniques in Software Engineering
http://drops.dagstuhl.de/opus/volltexte/2006/423

students. I explain the teaching context during which program transformation was taught. Then, I present the objectives, structure and content of the program transformation unit used for teaching and the different variants of this unit. Finally, I conclude by discussing the challenges faced during this experience and the lessons learned. By sharing these experiences, I hope the reader will find some guidance in developing and delivering educational material on the application of program transformation in software reengineering.

Teaching Context

In 2003, the Department of Computer Science at University of Leicester started its 1-year new taught M.Sc. of Software Engineering for the e-Economy program (renamed from 2005 as the M.Sc. of Advanced Software Engineering). The program consisted of two semesters of modules and 3-months summer project and thesis. In each semester the students have to take a mixture of mandatory and optional courses. In 2003/2004, the mandatory modules were:

- System Reengineering,
- Generative Development,
- Web Technologies,
- Advanced System Design, and
- two small modules: Seminar and Planning. These two combined have half the weight of one of the four above and serve the transferable skills of the M.Sc. program.

Additionally, students needed to choose two optional modules from a menu of software engineering and computer science modules.

The program is distinguished in offering full modules both on software system reengineering and generative development. The aim of the System Reengineering module is to educate students and train them on the realities of software industry, in terms of having to deal with and evolve legacy code-bases. It equips them with the knowledge and skills of how to deal with the past and transform it to the future. The module covers the important aspects of software reverse engineering and reengineering. The outline of the module covers the following topics.

- Introduction to Legacy Systems
- Introduction to Software Evolution, Maintenance and Reengineering
- Program Analysis
- Complexity and Maintainability Metrics
- Program Transformation
- Software Refactoring
- Web-enabling Legacy Systems
- Management Issues in Software Reengineering Projects

In 2004/2005, this module was assessed by a final exam (50%) and four pieces of coursework (50%). The coursework included a class test and 3 assignments. The first is an essay assignment, in which students write a group essay and give a presentation on a reengineering or reverse engineering topic that is not covered in the curriculum. Some example topics are: de-compilation, architecture recovery, reengineering of object-oriented systems and database reverse engineering and reengineering. The other two assignments are practical mini (or micro) projects on code transformation and refactoring.

Teaching Program Transformation

In this section we focus on the program transformation unit of this module by describing its objective, structure and content. This unit is 2 to 2.5 weeks long, i.e., 10 to 12.5 hours long. It assumes that the student needs 1.5 hours of self-study/homework for every hour of instruction or lab. The objective of teaching program transformation is to introduce the students to this important reengineering tool and get them to understand and try out what can be accomplished with it in the context of reengineering. In particular, this unit focuses on source-to-source code transformation. Two variants of this unit were offered, one in 2003/2004 and one in 2004/2005. They are described in the following.

Teaching Code Transformation Using TXL

In 2003/2004, TXL [2,3] was chosen to teach code transformation. TXL, Turing eXtender Language, is a programming language and rapid prototyping system specifically designed to support rule-based source-to-source transformation [2]. TXL is well supported and its Web page [3] is very useful, well maintained and frequently updated. After consultation with Jim Cordy and Filippo Ricca, I understood the suitable size and depth of the unit to suit its context. They also shared with me some of their well-prepared materials. The TXL unit consisted of 6 hours of lectures and tutorials on basic TXL, 3 hours of labs, a class test and an assignment.

This unit gave students a flavor of what program transformation (particularly source-to-source) is about and what it can accomplish. Students invested a lot of time and effort in learning TXL. By the end of the unit, students could write small TXL programs. For example, they could write a grammar for a language that only has variable declarations, assignments, 'if', 'case', 'while', 'goto' statements and labels. Then, they could write a TXL clone detector or a TXL program-restructuring (‘goto’ elimination) tool for programs in this language.

While no statistical data was collected, it was noted from an evaluation questionnaire at the end of the module and from discussions with the students that mature students with work experience appreciated and enjoyed the unit more than those who did not have previous work experience. These students had deeper understanding of what this technology could be used for and the potential applications of it.

Teaching Code Transformation Using L-CARE

In 2004/2005, code transformation was taught via a condensed 12-hours industrial tutorial, in one week. The tutorial was funded by Leg2NET project [4] and was part of the Transfer of Knowledge (TOK) component of this project. In this offering, an industry expert from ATX Software taught the tutorial on campus using ATX’s Legacy Computer Aided Reengineering Environment (Legacy-CARE or L-CARE) [11]. L-CARE is a commercial environment for legacy software reverse engineering and reengineering that can perform:

- Program analysis: Control and data flow graphs, program dependence graphs, slicing and code queries. All graphs are XML-based and can be queried with an extension of XPath.
- Rule-based mass program transformations.
- Documentation and visualisation
- Metrics

The program transformation unit, i.e., the industrial tutorial covered:

- Introduction to Reengineering and L-CARE Environment
- L-CARE Code Pattern Detection and Code Query Demo and Exercises
- L-CARE Code Transformation Demo and Exercises
- Industrial Case Studies on Applications of Code Pattern Detection and Code Transformation in L-CARE
- Slicing and Code Views in L-CARE
- Survey of Existing Reengineering Tools

The tutorial consisted of lectures, demos and lab sessions. Two big industrial case studies were demoed. The first was on using code pattern detection for code certification. In this case study, structural rules were written to query code graphs for non-compliance with an organization's code style manual. The application was used to enforce compliance with the manual before code goes to testing. In the second case study, rule-based code transformation was done to transform a huge Cobol system from persistent file storage to database storage. These case studies were very appreciated by the students and made them really appreciate the technology. This was clear from an evaluation questionnaire that was done at the end of the tutorial. Assessment was done by an assignment that included pattern detection and code transformation tasks.

Challenges, Questions and Lessons

In the course of teaching the program transformation unit two times, some challenges and questions were faced and some valuable lessons were learned. I elaborate on them combined in the following since through these challenges, lessons were learnt. They are not meant to be in a particular order.

Theory vs. practice. The first questions to ask before teaching program transformation in the context of software reengineering or in a different context is what are the learning objectives and timeframe of this unit and how does the unit fit in the rest of the module or course? Depending on the answer, one can decide whether a theory-oriented unit or a practice-oriented unit should be offered and how much subject should be covered. In my case, the focus was on practical and applied aspects and the relevant underpinning theory. To my knowledge, there is no textbook and very little ready-to-use educational material on the topic. Hence, considerable effort needs to be spent on deciding the appropriate material. However, if it is decided to teach applied program transformation and some tool(s) are selected, then it is natural to choose the literature related to the tool for preparing the unit material.

Which tool(s) to use? There are many research and commercial program transformation tools that support a wide range of transformation tasks. The two mentioned here, TXL and L-CARE, are just members of a bigger family. The problem of choosing a suitable tool is that hardly any of the available tools was designed for/used in teaching program transformation in a classroom, to my knowledge. Hence, often neither educational materials nor previous experiences in teaching using these tools exist. This means that for a newly used tool, materials (lectures, labs, assignments, projects, etc.) need to be prepared from scratch for the context it will be used in. But what is more difficult, at least in my experience, is judging how the learning curve of the students would be like with a given tool and what type and size of examples, exercises, and projects to give them.

Additionally, depending on the module content, other tools may also be used. Hence, it is better to use multi-purpose tools and use as few tools as possible. In 2003/2004, I taught students using a number of tools for various tasks in System Reengineering module. They used Imagix4D [5] for reverse engineering, visualization and metrics, CodeSurfer [6] for slicing, TXL [3] for code transformation and IntelliJ [7] for refactoring. Every time I introduced a new tool to use for the next 2 or 3 weeks, students reaction was: "Oh, no! Enough new tools, please please." It became overwhelming by the end of the module. I learned that picking a good or even the top tool for every unit is not always the best option. In 2004/2005, I tired introduce as few tools as possible by selecting multiple purpose tools. For code transformation, L-CARE was selected not only because of our collaboration with ATX Software, but also because it supports reverse engineering, visualization, slicing, metrics and program transformation. The only other tool that was used with L-CARE was a refactoring tool (students had the choice of using Eclipse [8] or IntelliJ [7]).

Exercises and projects: too small, too big. Without enough experience in teaching this topic, it is very hard to decide what size of programs students can handle in coursework. This applies not only to code transformation, but to refactoring, reverse engineering, etc. as well. In 2003/2004, I had to abandon a mini project for reverse engineering a small size open source software because it was way too much for the students than what I thought when I designed it. For program transformation, students had to do a small assignment using TXL. While this was enough work for the students and suitable for their assessment, it was just a toy example with no resemblance to reality. Again in 2004/2004, we had only small homework on program transformation using L-CARE on toy exercises. One good advice to judge the effort needed for an assignment or a project is to do it yourself in full, and then multiply the time spent by a suitable factor between 1.1 and 3, depending on how you see your speed compared to the students in solving the given problem.

Students Appreciation. One of the objectives of this module is to get students to understand and appreciate the complexity of changing legacy systems. Unfortunately in 2003/2004, students with no work experiences did not get this message well enough, especially when most of the examples, labs and assignments they had were toy examples. Students with industrial experience had deeper understanding and appreciation of the reality of software change. This was one of the reasons behind trying L-CARE in 2004/2005. In this offering, we had an industry expert on site and we had all examples and demos driven from industrial program transformation applications. Lab exercises and assignments also had a flavor of real applications. This helped achieving the unit objective.

Acknowledgment

The author acknowledges the financial support of Leg2NET project (From Legacy Systems to Services in the Net), which is a joint project between the Software Specification and Design Group at University of Leicester and ATX Software, funded by the European Commission under Marie Curie Industry-Academia Strategic Partnership Scheme (ToK-IAP), contract #3169 and led by professor José Luiz Fiadeiro. This support made it possible to host the industrial tutorial on program transformation offered for University of Leicester M.Sc. of Software Engineering for the e-Economy students in 2004/2005. The author acknowledges the effort of ATX Software in preparing and delivering the L-CARE tutorial and the hard work and dedication of Georgios Koutsoukos in preparing and teaching the tutorial. The author

acknowledges the advice and materials provided by Jim Cordy and Filippo Ricca in order to deliver a program transformation unit using TXL during the academic year 2003/2004.

Reference

[1] A. van Deursen, J. Favre, R. Koschke and J. Rilling, *Experiences in Teaching Software Evolution and Program Comprehension*. International Workshop on Program Comprehension (IWPC'03), p. 283, 2003.

[2] J. Cordy, T. Dean, A. Malton and K. Schneider, *Source Transformation in Software Engineering using the TXL Transformation System*. Special Issue on Source Code Analysis and Manipulation, Journal of Information and Software Technology 44(13), pp. 827-837, October 2002.

[3] TXL Web Page, http://www.txl.ca

[4] From Legacy Systems to Services in the Net, A Marie- Curie TOK-IAP Project (Leg2NET), http://www.cs.le.ac.uk/SoftSD/Leg2Net/

[5] Imagix4D, http://www.imagix.com/products/imagix4d.html

[6] CodeSurfer, http://www.grammatech.com/products/codesurfer/

[7] IntelliJ, http://www.jetbrains.com/idea/

[8] Eclipse, http://www.eclipse.org/

[9] I. Sommerville, *Software Engineering*, 6th Edition. Addison-Wesley, 2000.

[10] I. Sommerville, *Software Engineering*, 7th Edition. Addison-Wesley, 2004.

[11] Legacy-CARE, http://www.atxsoftware.com/?sec=products&it=48