Experience in Teaching a Software Reengineering Course

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

ABSTRACT
Software engineering curricula emphasize developing new software systems. Little attention is given to how to change and modernize existing systems, i.e., the theory and practice of software maintenance and reengineering. This paper presents the author's experience in teaching software reengineering in a masters-level course at University of Leicester, UK. It presents the course objectives, outline and the lessons learned. The main lessons are: first, there is a big shortage of educational materials for teaching software reengineering. Second, selecting the suitable materials (that balance theory and practice) and the right tool(s) for the level of students and depth of coverage required is a difficult task. Third, teaching reengineering using toy exercises and assignments does not convey the practical aspects of the subject. While, teaching with real, even small size, exercises and assignments, is almost infeasible. Getting the balance right requires careful consideration and experimentation. Finally, students understand and appreciate this topic much more if they have previous industrial experience and when they are presented with real industrial case studies.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Restructuring, reverse engineering, and reengineering.*

General Terms
Documentation, Human Factors.

Keywords
Software Reengineering, Software Engineering Education, Reengineering Course, Software Reengineering Education.

1. 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 a legacy from the past that they have to understand, admire, take care of and evolve. This requires knowledge and skills in the areas of program comprehension, evolution, maintenance, reverse engineering and reengineering. These areas have received a lot of attention from the research community, resulting in an increasing number of projects, conferences and workshops, e.g., WCRE, ICSM, CSMR, IWPC (now ICPC), IWPSE, WOOR and others. Unfortunately, software engineering (SE) curricula are significantly lagging behind in providing the necessary training on these areas, and particularly on the areas of reengineering and reverse engineering. Most of the time, students are trained on developing new small programs from scratch, but are not taught how to understand and change existing and large ones [1]. SE textbooks cover software change and evolution minimally, as a side topic. Compare, e.g., the 6th and 7th editions of Software Engineering by Sommerville. Chapters 26, 27 and 28 in the 6th edition [10], which are titled Legacy Systems, Software Change and Software Re-engineering, respectively, are reduced in the 7th edition to only Chapter 21: Software Evolution [11].

There is a need for more emphasis in SE undergraduate and graduate programs on the issue of software evolution and change. Students need to be educated on the theory and practice of software comprehension, maintenance and reengineering. They need to learn how to live with the monsters from the past and tame them. There is an equal need for packages of educational materials of different levels of depth for different curricula.

In this paper, I share my experience in teaching software reengineering to students on a one-year M.Sc. program on SE at University of Leicester, UK. First, I give the broader picture of the aims of the software reengineering course. Then, I explain the M.Sc. program context, within which reengineering is taught. Then, I present the structure and content of the course and its different variants. Finally, I present the lessons learned from this experience. By sharing my experience, I hope to attract the attention of SE educators to the importance of including software reengineering in their curricula and to give some guidance and help to those who are developing and delivering similar courses.

2. WHY TEAHCING REENIGNBEERING?
Software reengineering is "the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form" [12]. Reengineering is different from software maintenance, which is "the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment" [13]. But, the boundaries between reengineering and maintenance are not well-defined. While software maintenance is briefly covered in some SE curricula, reengineering is almost ignored.

2.1 The Reengineering Spectrum
Reengineering covers a very wide spectrum of activities as in Figure 1, including database reengineering [15], program transformation [16], design refactoring [17], user interface reengineering / Web-enabling [18] and supporting activities like risk management [19], costing [20] and guidelines [21] for reengineering projects. A reengineering activity usually consists

ICSE'06, May 20-28, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.

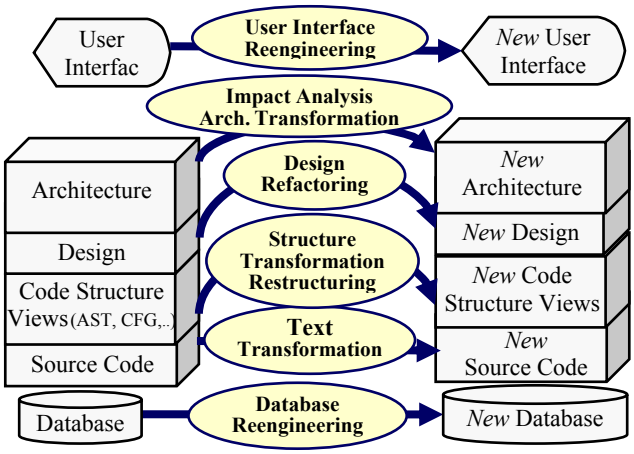

Figure 1. The Spectrum of Software Reengineering Activities

of two phases: The first is a reverse engineering phase in which the system is analyzed to identify its components and their relations and to create representations of the system at at a higher level of abstraction [12]. The second is a forward engineering phase in which the system is reconstituted in a new form. So, a reengineering course should address both phases in a balanced way. Reverse engineering also covers a wide range of activities.

2.2 Course Goals

Reengineering skills are survival skills for those who have to carry out software renovation and modernization projects. But, it is difficult to cover the entire reengineering and reverse engineering spectrum in a course. Hence, our course aims to give students basic knowledge and training on how to deal with the past and transform it to the future and give them a flavor of the important reverse engineering and reengineering activities applied on practical projects. Hopefully, they will expand what they learned and learn further techniques on their own whenever the need arises. Specifically, the goals of the course are:

- Understanding the problems and issues associated with legacy systems and some of the methods used to reverse engineer, comprehend, maintain, and reengineer them.
- Developing analytical and problem solving skills in reengineering legacy systems.
- Developing hands-on experience in reverse engineering and reengineering existing software systems.

The learning outcomes of the course were set as follows: By the end of the course, students should be able to:

- Understand software aging phenomenon, the challenges in renovating and maintaining legacy systems and the available methods for dealing with them
- Make reasoned decisions on which reengineering methods to apply for specific legacy system renovation tasks.
- Apply the methods learned to assess the situation of a small legacy system and decide a suitable reengineering strategy for it, in light of the objectives of the reengineering effort.
- Reverse engineer / reengineer moderate size legacy systems using some of the available commercial / research tools.

3. THE M.Sc. PROGRAM CONTEXT

In 2003, the Dept. of Computer Science at University of Leicester started its new one-year taught M.Sc. of SE for the e-Economy program (renamed in 2005 as the M.Sc. of Advanced SE). The program consists of two taught semesters and a three-months summer project with a thesis dissertation. In each semester, the students take a mixture of compulsory and optional courses. The M.Sc. degree weighs 90 ECTS credits [14]. The program's emphasis is on important and promising future trends in SE that are not so common in similar programs in the UK and worldwide. The taught part consists of core and supplementary courses. In 2005/2006, the core courses are: Personal and Group Skills, System Reengineering, Service-Oriented Architectures, Generative Development, Software Process Engineering and Advanced System Design. Students choose two optional courses from a menu of SE and computer science courses.

4. COURSE DEAILS

This section describes the details of the course: its contents, assessment and reading list. The official course name is CO7201: System Reengineering. The course was offered twice so far.

4.1 Course Outline

The course covers some techniques for analyzing, comprehending and measuring existing programs (reverse engineering) and some techniques for regenerating and modernizing existing systems (the forward engineering aspect of reengineering). The course outline includes: 1) an introduction to legacy systems, software aging, evolution, maintenance and reengineering, 2) program analysis and slicing, 3) complexity and maintainability metrics, 5) program transformation, 6) refactoring, 7) Web-enabling legacy systems and 8) management Issues in software reengineering.

The course was taught for 10 weeks, 5 hours a week: 3 hours lectures, 1 hour lab and 1 hour problem solving class. Sections 4.2 and 4.3 provides the specs of the two versions offered so far.

4.2 The First Version (2003/2004)

In 2003/2004, the course was taught for the first time. The course weight is 150 hours in University of Leicester standards, divided to 60 hours of lectures, labs and problem classes and 90 hours of private study. Suitable tools were selected for the course, each for a reengineering or reverse engineering task. These were:

- Imagix4D for program analysis and complexity metrics. [5]
- CodeSurfer for program slicing. [6]
- TXL for program transformation. [3]
- IntelliJ for program refactoring. [7]

The course was assessed by coursework (50%) and a final exam (50%). The coursework consisted of 3 quizzes, 3 practical and 1 theoretical assignments and a project. Rough estimates of how long the assignments and project will take were made. Theoretical materials were presented in the lectures. Further discussions and paper training of these materials were done in the problem classes. Practical training on the relevant tool was done in the labs and finally the assignments and the project allowed the students to practice and expand what they learned and reinforce their reengineering problem solving skills by solving mini case studies. The practical assignments were on: 1) calculating various slices and chops for a small system using CodeSurfer [6], 2) refactoring a vending machine simulator program manually and using IntelliJ

[7] and 3) writing a clone detector using TXL [3] for a simple language that has assignment, declaration, 'if', 'case', 'while' and 'goto' statements and labels.

The project included picking an unfamiliar C/C++ system after approval from the lecturer or choosing the system he proposed, Indent [2]. Then performing the following tasks on it:

- Reverse engineering, comprehension and architecture recovery of the subject system using the tools used in the course and manual analysis if needed.
- Producing technical documentation for the system.
- Performing one reengineering task on the system, either Web-enabling or transforming to another language or refactoring.

Overall, the course went well, with a few pitfalls. First, the estimation of the time needed for the project was grossly wrong due to: 1) my limited experience in teaching reengineering at the time and 2) the uncertainty and uniqueness of reengineering projects. Consequently, the project had to be scrapped halfway through the course and its mark was redistributed.

Second, personal communications with the students and the end-of-course questionnaire showed that the number of tools used was too many. My initial assumption that every two weeks, new material will be introduced and the relevant tool will be introduced in the lab and then used for an assignment proved to be too much. Instead of picking a very good tool for each reengineering topic, it is better to use the same tool for as many tasks as possible even if it is not the best. This relives the students form the burden of learning a new tool every two weeks.

On the positive side, students' evaluation showed that they liked and benefited from the course and that the practical assignments were very helpful in applying what they learned. They also said that the class tests forced them to study regularly and keep up to date. Most of the students did quite well on the course.

4.3 The Second Version (2004/2005)

In this version, the experience of 2003/2004 enhanced the course and students' learning. The outline remained the same with significant changes to the details of some topics and to the coursework. First, the number of tools used was reduced to two. We used L-CARE reengineering environment [9] for program transformation and analysis and slicing. Students had the choice to use IntelliJ [7] or Eclipse [8] for refactoring. This proved to be much more practical and less stressful to students. Second, we integrated an industrial tutorial on "Software Reengineering for Real" in the course, supported by Leg2Net project [4], which is a transfer of knowledge project between University of Leicester, UK, and ATX Software, Portugal. For demos, labs and assignments, L-CARE was used [9]. The tutorial outline was:

1. Reengineering Overview and Market.
2. Reengineering Scenarios Found in Practice.
    - Software Understanding Scenario.
    - Software Transformation Scenario(s).
    - Software Quality Assurance Scenario.
3. Introduction to L-CARE Environment.
4. XML Representation of Code.
5. Querying XML Code Representation with XPath.
6. Code Patterns Detection and Its Industrial Applications.
7. Code Transformation and Its Applications.
8. COBOL Quick Tutorial and COBOL Transformation in Industry.
9. Slicing, Code Views and Metrics in L-CARE.
10. Survey of Existing Source Code Reengineering Tools.

The tutorial was 12 hours, divided between presentations / demos and labs. It was successful and served a few purposes:

- It covered the practical training on several topics using one tool (program analysis, program complexity and metrics and program transformation).
- It gave the students a feeling of the reality of legacy systems and software reengineering.
- It presented to the students real industrial case studies with a small-scale case study to solve as an assignment. Research shows that teaching with industrial case studies is very beneficial to students [23].

The last change was redesigning the coursework to suit the time available to students and to broaden the course spectrum by getting the students to explore further topics on their own. The assessment consisted of 2 class tests and 3 assignments. The first assignment was a group research report and presentation on a topic not covered in the class, e.g., de-compilation, architecture recovery, database reverse engineering and reengineering, the role of XML in reverse engineering and reengineering, etc. The second assignment was a small case study on code pattern detection and transformation using L-CARE. The third assignment was a refactoring case study where students were asked to detect the bad smells in a vending machine simulator program, refactor it and write a report on their work. The students had the choice of using IntelliJ [7] or Eclipse [8] for refactoring.

An evaluation survey showed that students' satisfaction was higher than the previous year and that they benefited from and liked the industrial tutorial very much. However, they still regarded the course load as too much.

4.4 Reading List

When I started preparing the course, I found no textbook on software reengineering. So, I assembled a reading list from the available resources. It included: 1) Introductory articles on legacy systems, software aging, reengineering, etc., 2) Technical papers on specific techniques, and 3) User manuals for the tools used

5. LESSONS LEARNED

Valuable lessons were learned from the author's experience in teaching this course and from students' feedback. A survey was conducted at the beginning of the course to collect information about students' backgrounds and another one was done at the end of the course to evaluate it. The following gives my reflections and lessons learned, but without detailed statistics due to space limit:

1. Few SE programs cover the subject of software change and evolution. Fewer courses cover some aspects of software reengineering. When I started preparing my course, my search could not find any available courses on software reengineering to use for guidance. The subject is underrepresented in SE curricula. Students need education and training on how to deal with and evolve legacy code-bases.
2. Very little educational materials are available for teaching reengineering. While very significant achievements took place in reengineering research and in developing industrial tools,

most research materials and tools are not meant for education and require huge effort to adapt to an educational setting. There is a great need for tried training and educational packages for reengineering, similar to the LAN simulator refactoring lab [22], for example.

3. Very little pedagogic research was done on teaching and learning software change and evolution. The proceedings of the Conference on SE Education and Training in the last five years had only two papers on teaching reengineering-related topics. Compare this with research on teaching software process or on SE student projects. This shows that the subject is grossly under-researched as well.
4. Considering the wide range of reengineering activities, one can shape the course according to his/her and his/her students' interests and still serve its main objectives. But no matter what topics are included, it is important to cover practical tools and some industrial case studies.
5. It is important to use as few tools as possible. As my experience showed, it is overwhelming to students to use a new tool for every new reengineering activity introduced. It is better to use multi-purpose tools that can serve more than one topic. When picking a tool, one should carefully consider its learning curve. Heavyweight industrial tools have a slow learning curve and can be infeasible to use in teaching.
6. A tutorial taught by an industry expert proved to be very successful in teaching reengineering. It exposed the students to real industrial case studies and trained them on an industrial tool. Evaluation surveys showed that the vast majority of students liked and benefited from this tutorial very much.
7. Teaching theoretical aspects of reengineering must be accompanied by considerable practice via projects and assignments. But one should carefully estimate their time and effort. Mature metrics exist for estimating size and effort in new software development. But, in reengineering, every project is almost unique and one does not know what s/he is up to until s/he starts tackling the target system.
8. Evaluation surveys and marks showed that students who joined the M.Sc. program with past industrial experience, benefited from and appreciated the course the most. They related the material with their past experiences.

6. CONCLUDING REMARKS

This paper presented my experience in teaching software reengineering for students of M.Sc. of Advanced SE at University of Leicester. Overall, it was a successful experience and the goals of the course were achieved. It was not possible to cover all aspects of software reengineering in the course. So, only the key techniques were covered via lectures, labs, readings, an industrial tutorial and assignments. Most students reported that the course was interesting and beneficial. They gained confidence in how to tackle and renovate legacy systems and code-bases written by others.

In the future, I will work on developing new lab materials, case studies and project ideas and use the past data to better estimate the time and effort needed for assignments and projects.

7. ACKNOWLEDGMENTS

The author acknowledges the financial support of Leg2NET project [4] and the effort of ATX Software, especially Georgios Koutsoukos, in preparing/delivering "Software Reengineering for Real" tutorial. The author thanks Jim Cordy and Filippo Ricca for the TXL teaching materials they provided and thanks Imagix Corporation, GrammaTech, Inc. and JetBrains for their generous contributions of educational and free licenses of their products.

8. REFERENCES

[1] A. van Deursen, J. Favre, R. Koschke and J. Rilling, *Experiences in Teaching Software Evolution and Program Comprehension*. Int. Wkshop on Prog. Comp., p. 283, 2003.

[2] *Indent*, An Open Source GNU Program for Beautifying C Programs. http://www.xs4all.nl/~carlo17/indent/

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

[4] *From Legacy Systems to Services in the Net (Leg2Net).* A Marie-Curie ToK-IAP Project, Contract #3169. 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] JetBrains, *IntelliJ*. http://www.jetbrains.com/idea/

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

[9] ATX, *Legacy Computer Aided Reengineering Environment, L-CARE*. http://www.atxsoftware.com/

[10] I. Sommerville, Software Engineering, 6th Edition, 2000.

[11] I. Sommerville, *Software Engineering*, 7th Edition, 2004.

[12] E. Chikofsky and J. Cross, *Reverse Engineering and Design Recovery - a Taxonomy*. IEEE Software, Jan. 1990

[13] IEEE, ANSI/IEEE Standard 729-1983 and IEEE Standard 1219-1998. In IEEE Software Eng. Standards, IEEE, 1987

[14] *ECTS - European Credit Transfer and Accumulation System.* http://europa.eu.int/comm/education/programmes/socrates/ects/index_en.html

[15] *A General Meta-model for Data-centered Application Reengineering*. A Dagstuhl Seminar on Interoperability of Reengineering Tools (No. 01041), 2005.

[16] *Transformation Techniques in Software Engineering*. A Dagstuhl Seminar (No. 05161), 2001.

[17] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Obj. Tech. Series, Addison-Wesley Longman, 1999.

[18] R. Kapoor, *Device-Retargetable User Interface Reengineering Using XML*. TR TR01-11, Dept. of Computing Science, Uni. of Alberta, 2001.

[19] J. Bergey, D. Smith, S. Tilley, N. Weiderman and S. Woods, *Why Reengineering Projects Fail*. TR CMU/SEI-99-TR-010, SE Institute, Carnegie Mellon Uni., 1999.

[20] H. Sneed, Estimating the Costs of a Reengineering Project. Working Conf. on Reverse Engineering (WCRE), 2005.

[21] J. Bergey, D. Smith and N. Weiderman, *DoD Legacy System Migration Guidelines*. TR CMU/SEI-99-TN-01, SE Institute, Carnegie Mellon Uni., 1999.

[22] S. Demeyer et al., *The LAN-simulation: A Refactoring Teaching Example*. Int. Workshop on Principles of Software Evolution (IWPSE), 2005.

[23] J. Krone, D. Juedes and M. Sitharam, *When Theory Meets Practice: Enriching the CS Curriculum through Industrial Case Studies*. Conf. on SE Edu. & Training, 207-214, 2002.