Experiences in Teaching Software Evolution and Program Comprehension

Arie van Deursen*
CWI and Delft University of Technology
The Netherlands

Jean-Marie Favre†
University of Grenoble, France

Rainer Koschke‡
University of Stuttgart, Germany

Juergen Rilling§
Concordia University, Canada

Abstract

A large amount of research is devoted to software evolution and program understanding, but these topics are often neglected in software engineering curricula. The CCSE initiative from IEEE/ACM proposes guidelines for the future. At the ICSM'02 conference, a panel was held to establish what should be taught in the future. This working session focuses on what is being taught and what will be taught in the near future. The goals include (1) to share experiences in teaching software evolution and program understanding, (2) to establish the state of the practice and (3) to identify future directions.

1. Introduction

Understanding and evolving a large software product are known as very challenging tasks. To address these issues a large amount of research work has been lead over the last decades giving rise to conferences such as IWPC, ICSM, WCRE, CSMR and IWPSE and to the creation of numerous research projects and networks.

Despite of this large body of research, the impact on software industry practices has been however quite limited. Understanding large software is still a craft. While all practitioners face this issue, they often consider this as a dark part of software engineering: performing these tasks without any kind of methods or tools still seem natural.

Although some tools are available, tool adoption is an important issue. For instance, commercial reverse engineering tools produce various kinds of output, but software engineers usually do not know how to interpret and use these pictures and reports.

The importance of teaching software evolution and program comprehension has been recognized in the last years. International organizations such as IEEE/ACM are working on the definition of guidelines for a software engineering undergraduate curriculum [4]. The last ICSM hosted a panel entitled "How Should Software Evolution and Maintenance a Taught?" [1]. The objective of the present working session is to focus on the current state of the practice by answering the following questions "what is being taught currently?" and "what will be taught in a near future ?". In other words, the session will focus on sharing actual experiences in teaching software evolution and program comprehension. To goal is to establish a repository of teaching resources, to identify the lessons learned and future directions.

We invite participation by

- academics who have experience teaching such courses,
- academics who expect to teach such a course in the near future,
- tool vendors and practitioners who teach professional seminars on software evolution.

2. Teaching Program Comprehension and Software Evolution

Most of the time, students are trained in developing very small programs starting from scratch. This approach is really misleading since most students learn to believe that software engineering is just about developing brand new software. In fact many students will be involved in evolution-related activities after completion of their studies.

Students learn how to write new programs but they are not taught how to read and change existing and large ones.

The scale of the program is obviously an important factor to consider. When focusing on small programs, the traditional exercise / solution cycles work well. Increasing the size of the

*http://www.cwi.nl/~arie/
†http://www-adele.imag.fr/~jmfavre
‡http://www.informatik.uni-stuttgart.de/ifi/ps/rainer/index.html
§http://www.cs.concordia.ca/~faculty/rilling/

Proceedings of the 11 th IEEE International Workshop on Program Comprehension (IWPC'03)
1092-8138/03 $17.00 © 2003 IEEE

software to be developed could be done by focusing on team development. To put emphasis on integration and reuse many software engineering courses now include also include task involving the understanding of large and complex APIs. As a result, various university successfully teach how to develop medium scale systems.

However, evolving and understanding large existing software includes quite different activities such as recovering/understanding the actual architecture of the system, understanding some part of its source code and documentation (may be in some programming language that the student had not studied before). Including a new feature in such software then requires to perform impact analysis, regression testing, etc. Teaching these issues is quite challenging.

The SEEK initiative identifies different topics in the "Software Evolution" knowledge area [4]:

Evolution Processes basic concepts of evolution and maintenance, relationships between evolving entities, theories, cost models, planning.

Evolution Activities: Working with legacy systems, program comprehension and reverse engineering, system and process re-engineering, impact analysis, migration (technical and business), refactoring, program transformation, and data reverse engineering.

The list could be refined. In the context of this session, we are in particular interested in establishing how these topics are covered by the various existing courses, and how this to turn this wish list into practice. In particular we seek for

- examples of effective exercises, labs, and demonstrations;
- examples of effective evaluation tools and grading techniques.

Questions to be addressed include:

- What topics are currently covered?
- Which guinea pig systems are being used? What size of software is appropriate? Is it convenient and possible to reuse the same system over the years? What level of understanding of these systems should the teacher have?
- Are classical small scale exercise usable?
- Which set of slides, papers and books are currently used?
- Which resource on the web (e.g., Reengineering Wiki [2], REportal [3]) are used?
- Which tools (commercial tools / research prototypes / web REportal) are used?
- What kind of evaluation work best? How should a program understanding task be graded?
- What about the understanding of source code of unknown programming languages?

Obviously the answers to these questions largely depend on the level and the target of the curricula (research oriented vs. industry oriented), but the focus on the session is more to study existing courses and material rather than to define appropriate curricula.

3. Format of the Working Session

The first part will discuss the state of the practice by summarizing selected courses and topics, as well as relevant resources available on the web. All participants can contribute to this overview.

The second part of the panel will be highly interactive. The goal is to collect information and share results about best and worst experiences, and promising ways to teach program comprehension in the near future.

In order to encourage interaction, participants will be asked to write short responses to specific statements or questions on index cards. The answers will be collected, and categorized in a group discussion.

4. Results

Information on relevant program comprehension courses and relevant resources will be made available via the pages devoted to teaching[1]the Reengineering Wiki [2]. We anticipate to make active notes on the outcomes of the session, leading to a report on the current state of practice in teaching program comprehension and software engineering.

References

[1] A. van Deursen, T. Lethbridge, and P. Stevens. How should software maintenance and evolution be taught? In *Proceedings of the International Conference on Software Maintenance (ICSM 2002)*. IEEE Computer Society, 2002.

[2] A. van Deursen and E. Visser. The reengineering wiki. In *Proceedings 6th European Conference on Software Maintenance and Reengineering (CSMR).*, pages 217–220. IEEE Computer Society, 2002.

[3] S. Mancoridis, T. S. Souder, E. R. Gansner, and J. L. Korn. REportal: A web-based portal site for reverse engineering. In *Eighth Working Conference on Reverse Engineering (WCRE'01)*, pages 221–229. IEEE Computer Society, 2001.

[4] A. E. K. Sobel, editor. *Computing Curricula — Software Engineering Volume; Second Draft of the Software Engineering Knowledge (SEEK)*. ACM/IEEE, December 2002.

[1] See `http://www.program-transformation.org/twiki/bin/view/Transform/TeachingSoftwareEvolution`