

# How Should Software Evolution and Maintenance be Taught?

Arie van Deursen  
CWI, P.O. Box 94079  
1090 GB Amsterdam, The Netherlands  
<http://www.cwi.nl/~arie/>

Timothy C. Lethbridge  
University of Ottawa  
Ottawa, Canada, K1N 6N5  
<http://www.site.uottawa.ca/~tcl/>

Perdita Stevens  
University of Edinburgh  
United Kingdom  
<http://www.dcs.ed.ac.uk/home/pxs/>

## Abstract

*The IEEE/ACM CCSE initiative to propose guidelines for an undergraduate program in software engineering provides an opportunity to rethink the role of software maintenance and evolution in software engineering curricula. The purpose of this panel is to share experiences and discuss novel ways in which evolution and maintenance can be incorporated in an undergraduate software engineering curriculum.*

## 1. Introduction

Although software maintenance and evolution account for the majority of the costs involved in a software system's life cycle, the amount of time spent on these topics in a typical computer science or software engineering curriculum appears to be negligible. This is even more surprising given the likelihood that software engineering students will be involved in maintenance-related activities after completion of their studies.

Discussions on the role of software maintenance and evolution in computing curricula have become particularly relevant in the context of the IEEE/ACM *Computing Curricula Software Engineering* (CCSE)<sup>1</sup> initiative in which guidelines are being developed for an undergraduate program in software engineering. Work in the CCSE committees is well underway, and divided into a number of *knowledge areas*, one of which covers the field of *software evolution and maintenance*.

The purpose of this panel is to share experiences and discuss novel ways in which evolution and maintenance can be incorporated in an undergraduate software engineering curriculum.

---

<sup>1</sup>See <http://sites.computer.org/ccse/>

## 2. The work plan for CCSE

The IEEE and ACM have, for many years, collaborated to produce curriculum recommendations for computer science. The most recent such document was released in late 2001 and is available at <http://www.computer.org/education/cc2001/>

The computer science volume is, however, only one of four volumes that are to be part of the overall Computing Curriculum document. The others are to cover Software Engineering, Computer Engineering and Computer Science. From this point forwards, the Computing Curriculum document is to be a living document, regularly updated.

The software engineering volume (CCSE) has been under development since mid-2001. Its development is managed by a steering committee headed by Richard LeBlanc and Susan Mengel. There are two subcommittees, one devoted to what knowledge should be taught, and the other to the pedagogy, i.e. *how* the knowledge should be taught. Together, these committees have about 100 listed volunteers, although about half that number have been active as of Summer 2002.

The knowledge area committee was divided into subcommittees, each looking into a particular area. One such subcommittee was originally called Software Maintenance, but was later changed to Software Evolution and Maintenance. Knowledge area committee work was performed online and at two workshops in the spring of 2002. The resulting work, called Software Engineering Education Knowledge (SEEK) was considered ready for a first formal review by software engineering experts in July 2002. Following this process, in late summer 2002, the revised document was due to be opened for a public review.

SWEBOK was one of the inputs to SEEK, however SEEK is not duplicating the work of SWEBOK because SEEK describes what should be taught to undergraduates,

as opposed to what practitioners should know after four years of work. Some material in SEEK, e.g. professional ethics and discrete math, is not listed in SWEBOK; similarly, SWEBOK contains knowledge that would be too advanced to teach to undergraduates.

The pedagogy group will start to formulate a curriculum based on the SEEK once it stabilizes.

### 3. Software Evolution and Maintenance in CCSE

As of the time of writing (mid summer 2002) SEEK was still being reviewed by experts and was subject to such extensive revision that it was deemed not prudent to make it public. Nevertheless, the following is a rough list of what is anticipated will be in the core of the evolution and maintenance area. By core, we mean that the knowledge will be required to be taught to all undergraduate software engineering students:

1. Configuration management: revision control, release management, builds etc.
2. Maintenance process models
3. Reverse engineering, program comprehension, impact analysis, reengineering, refactoring, migration, regression testing etc.
4. Impact Analysis: change propagation, modularization, program analysis.

Other core topics related to evolution and maintenance, e.g. cost estimation, can be found in other knowledge areas, such as project management.

There will also be some optional topics in SEEK that could be used to build elective courses that cover evolution and maintenance more deeply.

We anticipate being able to make an up-to-date version of SEEK available to attendees of ICSM.

### 4. Panel Questions

Questions to be addressed by the panelists and audience members include (but are not limited to):

- What topics should be listed as 'core' in the software evolution and maintenance area of SEEK.
- How can we ensure that the curriculum prepares students well for industrial software evolution practice?
- How early in the curriculum can and should maintenance be addressed?

- What are key learning objectives for software maintenance courses?
- How can we devise suitable project work for the learning objectives?
- What percentage of a software engineering curriculum should be devoted to topics specific to evolution and maintenance?

### 5. The Goldfish Bowl Format

In order to encourage audience participation, the panel will follow the *goldfish bowl* format<sup>2</sup> as also used in, for example, the *object technology* and *extreme programming* communities.

A Goldfish Bowl is a process designed to manage an open discussion in a large group of people. There are a small number of seats in the middle and only the occupants of those seats may talk. Speakers who have run out of things to say, or would like to take a break, leave their seat and anyone else from the audience can take their place; speakers can return later when another chair opens up. There is also a moderator who introduces the session and can evict speakers if the discussion needs to be moved on. This Goldfish Bowl will be seeded by the initial panelists below; each will give a brief (5 minutes) talk to describe their experiences with the topic, then the session will be opened up to the audience.

The purpose of the session is to share the experiences of the whole community. The advantage of a Goldfish Bowl is that it allows everyone in the audience to contribute without always referring to a panel. It also avoids a potential weakness of panel sessions that, once the initial positions have been taken, the rest of the session is predictable.

### 6. The Panelists

**Gerardo Canfora** is a full professor of Computer Science at the Faculty of Engineering and the Director of the Research Centre on Software Technology (RCOST) of the University of Sannio in Benevento, Italy. His research interests include software maintenance, program comprehension, reverse engineering, reuse, reengineering, migration, workflow management, and knowledge and content management. He is the Project Work Coordinator in the Master Program on Software Technologies, an innovative training program developed within a University-Industry partnership. He has taken an active role in the University-Industry co-design of new training methodologies for high level professional figures and industrial researchers with

---

<sup>2</sup>Thanks to Steve Freeman for providing the goldfish bowl description.

competencies in the area of design, development, management, and evolution of software systems and services. The master course, this year in the 3rd edition, is delivered by the University of Sannio in collaboration with several industrial partners (e.g. STMicroelectronics, Ericsson Lab Italy, SchlumbergerSema, etc.). The innovative formula is the fact that students work in groups on realistic Project Work concerned with themes of interest of the industrial partners, integrating technological and managerial competencies.

**Jim Cordy** is Professor and Director of the School of Computing at Queen's University in Kingston, Canada. From 1995-2000 he was vice president and chief research scientist at Legasys Corporation, a company specializing in industrial software system analysis, maintenance and migration tasks. He is a member of IFIP Working Group 2.4 (Software Implementation Technology) and the author or co-author of numerous research contributions in software implementation and maintenance. Presently he serves as program co-chair of SCAM'02, the IEEE 2nd International Workshop of Source Code Analysis and Manipulation, and as industrial applications co-chair of ICSM'02. Dr. Cordy has more than 18 years experience in designing and teaching undergraduate and graduate courses in software science and engineering.

**Arie van Deursen** is a member of staff of CWI, the Dutch national research institute in mathematics and computer science. He teaches software engineering to computer science undergraduates at the University of Amsterdam. His software engineering course is designed to cover software evolution as well as software testing as one of the first topics. He is chair of the CCSE software evolution and maintenance knowledge area, and compiled the initial version of the software evolution units in CCSE. For the University of Amsterdam he has designed a *professional masters* program in software engineering: in this one year masters course software evolution is the first major topic covered.

**Timothy C. Lethbridge** is an associate professor at the University of Ottawa. He researches software reverse engineering and visualization, as well as software engineering education. He is co-chair of the Pedagogy Committee of CCSE and was the lead developer of the University of Ottawa's undergraduate software engineering program (one of the first in North America to be accredited). He is also lead author of the McGraw-Hill textbook "Object-Oriented Software Engineering: Practical Software Development using UML and Java".

**Hausi Müller** is a professor in the Computer Science at the University of Victoria, Canada, where he has been leading the process of establishing an undergraduate software

engineering program. He has conducted research in reverse engineering for many years. He has also held several leadership roles in software engineering, most notably as general chair of ICSE 2001. He was one of the expert reviewers in the first round of CCSE reviewing.