Construction by configuration: A new challenge for software engineering education

Ian Sommerville, Computing Dept., Lancaster University

1. Introduction

Software reuse has been the subject of software engineering research for more than 20 years. Although there are still problems to be overcome, reuse is one of the major success stories of software engineering and software development with reuse has become the dominant approach to the development of business software systems. However, although component-based software engineering [1] is starting to have a significant impact, the majority of this reuse is not the reuse of software components written in conventional programming languages such as Java or C++. Rather, it is the reuse of large-scale modules in ERP systems such as SAP/R3 or the reuse of generic software systems, designed for a particular domain and configured for the needs of specific customers.

If we look at the teaching of software engineering, however (and, it must be said, the presentation of the subject in software engineering textbooks), this approach to development is rarely mentioned. Mili's book on software reuse [2], for example, devotes 20 pages out of 600 to COTS-based reuse and does not mention problems of configuring COTS for specific customer requirements. The focus of teaching is primarily on the engineering of systems using conventional programming languages. In the ACM/IEEE proposals for curricula in software engineering [3], software reuse and COTS are mentioned briefly but are given very little emphasis and, in general, they are always considered to be part of something else, such as software evolution, rather than topics in their own right.

By ignoring this economically significant to software development, I believe that we are doing our students a disservice. Of course, students have to understand conventional software development practice and, of course, this is a major challenge in its own right. However, by focusing exclusively on this topic we do not equip our students with an understanding of the realities of much software development in the 21st century and we exacerbate the problem that many employers of our students do not believe that student skills match their real needs.

In this paper, I discuss the principles of 'construction by configuration' and the very real difficulties of teaching those principles in a university environment. I then go on to suggest how we, as educators, might address these problems and so broaden the scope of software engineering education.

2. Construction by configuration

The reuse of off-the-shelf software systems, in one form or another, is a reality for the majority of businesses. The most obvious manifestation of this is in the use of ERP systems such as the widely used SAP/R3 system where businesses buy a set of generic modules from the system provider and, with the help of consultants, tailor and adapt these modules to fit their specific business. Other widely used approaches are in the adoption of vertically integrated systems (e.g. a system for handling the admission of patients to a hospital) which are specialised for use in particular environments.

The characteristics of software engineering for such systems is that the focus of development moves away from the design of the system architecture and the production of a system implementation to the understanding of the requirements of a particular setting, the configuration of the system to reflect these requirements and the testing of the system in its operational environment. There is rarely a detailed system specification and the configuration of the system is often an ad hoc and sometimes even an undocumented process.

Many problems can arise in this process ranging from difficulties in understanding how different configurations affect system requirements for performance, reliability etc., unanticipated interactions between the configurations of different system features, difficulties in debugging configurations, and so on. I believe that these problems are just as significant for software engineering than the problems that arise in the development of conventional software systems.

3. Problems in teaching 'construction by configuration'

The fundamental difficulty that we face in teaching this (fairly) new paradigm of software development is that it has been virtually ignored by software engineering theorists and researchers. There is no public body of knowledge on good practice in system configuration and no principles or theory on which courses can be based. Consequently, there are no textbooks on the topic except very specific guides to individual systems. All of this means that instructors are understandably reluctant to incorporate this development method in their courses.

This is not the only problem. There are many other new developments that compete for inclusion in software engineering courses – security engineering, service-centric systems, aspect-oriented software development, for example, along with pressures to develop skills in students such as web page design and programming. Finding a place in the curriculum and, realistically, developing the skills of teachers are major issues.

4. Teaching 'construction by configuration'?

It is clearly impossible to include all new developments in a university course and the 'normal' response of educators is that their job is to teach principles and the foundations of a discipline and thus equip students to learn about new developments themselves. However, this approach only really works when we operate within a single paradigm. For example, if we teach the principles of programming a Von Neumann machine, then students should be able to use these as a basis for learning any conventional programming language. However, when we step outside the boundaries of these principles (e.g. say we expect students to learn an AI language such as Prolog), then it becomes extremely difficult for students to apply their knowledge to new things.

I argue that 'construction by configuration' is a different paradigm of software development and an understanding of generic principles of programming (such as the notion of sequence, loops and conditionals, data typing, etc.) is not enough for students to teach themselves this approach. Rather, students have to learn how to understand the characteristics and limitations of a specific application and how to reflect the needs of an operational environment in that application. At this stage, as we are lacking principles, I believe what we should be doing is making students aware of this approach to software development and giving them some experience of the problems that can arise.

At first sight, this appears to be a difficult problem. ERP systems are large and expensive and their purchase cannot be justified for teaching alone. Vertically integrated systems, such as hospital systems, require specialised domain knowledge that university teachers cannot be expected to develop. Fortunately, however, most of us and most students have a program on our computers that is an excellent vehicle for illustrating the problems and benefits of construction by configuration – Microsoft Excel.

Excel is an excellent example of a generic system that can be configured to construct specific systems that meet a particular need. Its characteristics reflect the differences between conventional programming and construction by configuration namely:

1. A programming model that is different from the model reflected in conventional programming languages.
2. The ability to construct quite different systems from a single generic base, where these different systems should reflect the specific business purpose that the application is designed to support.

3. A rather ad hoc approach to configuration, different ways to configure the system and a lack of facilities to make the configuration visible.

I argue, therefore, that as well as setting programming assignments in conventional programming languages, we should also be teaching construction by configuration by developing Excel-based projects and incorporating these in a more general course on the use of COTS-based systems. Such exercises would, I believe, teach students about the practicalities of software engineering in the 21st century and would help them to understand that the majority of the world's software is not developed in Java using an object-oriented approach.

5 References

[1] Szyperski, C. 2002. *Component Software: Beyond Object-Oriented Programming*. Harlow, UK: Addison-Wesley.

[2] Mili, H., Mili, A., Yacoub, S. and Addy, E. 2002. *Reuse-based software engineering: Techniques, Organization and Controls*. New York: Wiley Interscience.

[3] IEEE/ACM. 2004. *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. http://sites.computer.org/ccse/.