Software Configuration Management Problems and Solutions to Software Variability Management

Lars Bendix
Department of Computer Science
Lund Institute of Technology
Sweden
bendix@cs.lth.se

Abstract

These days more and more software is produced as product families. Products that have a lot in common, but all the same vary slightly in one or more aspects. Developing and maintaining these product families is a complex task. Software configuration management (SCM) can, in general, support the development and evolution of one single software product and to some degree also supports the concept of variants. It would be interesting to explore to what degree SCM already has solutions to some of the problems of product families and what are the problems where SCM has to invent new techniques to support software variability management.

1. Introduction

Software product families are becoming more and more common. There are many benefits from making and selling basically the same product in many slightly different variations. You can amortise your investment on a greater number of sold products. You can satisfy your customers' desire for specialised products at a reasonable cost. Your product will be able to run on many platforms and support many languages in a globalised market. You will be able to reuse parts from existing products allowing a faster time-to-market time.

However, there is also a down side to software product families. They are very complex to develop and even more complex to maintain and further evolve. And because of the high development costs and popularity of such products they tend have long lives. In order to balance the costs of the production with the benefits from increased sales of software product families, we need to be able to manage the complexity of software variability.

In my opinion, Software Configuration Management (SCM) has a major role to play when it comes to handling the complexities of software variability. SCM already has a fundamental role in supporting the development and evolution of single product software. If we were able to distinguish exactly how product families differ from single product software, we might discover that SCM can provide – or develop – techniques to support product families too.

In the following, I will first briefly outline the most important concepts and principles of SCM relevant to product families, after which I describe how I look at software product families. Then I will detail where I see previous relations between SCM and SVM – and finally state my position on what could be some of the discussion points at the workshop regarding the relationship between SCM and Software Variability Management (SVM).

2. Software Configuration Management

For the past two decades my main interest has been SCM. It is a discipline that spans a wide spectrum of functionality, ranging from computer supported co-operative work, that supports the developers in their tasks, to product data management, that enables the company to document the exact composition of their products.

SCM is the control of the evolution of complex software systems. Traditionally it is said to cover the life-cycle phases from coding to retirement. However, you can apply SCM principles and techniques to everything from your requirements specifications through design documents to documentation.

One of the core concepts of SCM is the management of a repository of components. Everything that the developers produce goes into this repository where it remains forever, such that it can be retrieved again at any time. When changes are made to something in the repository, they are not made directly to the component, but rather to a copy and the modified copy is then added

to the repository creating a new version of that component. This means that we can have many versions of the same component.

Another core concept is that of a product model. This product model describes the structure of the product by relating the components in a dependency graph. Furthermore, the product model also contains information about which actions to perform to build the product from its components. This means that once this product model has been described it is possible to automatically build the product.

When we put these two concepts together we get a slightly more complicated picture. The fact that the repository might contain more that one version of a given component means that there is a choice between which version to use for each component when we build the product. This gives rise to the concept of a generic product model that gives only the structure. This generic model is then bound to particular versions of components using the repository and a selection profile describing which version to use for which components.

These are problems where SCM has solutions that are well understood and mature. A naive and simplistic view at SVM would be to look at variants the same way as versions and use these already established solutions. However, variants do differ from versions so such an approach has strong limitations.

3. Software Product Families

Unfortunately I am not yet an expert on the subject of product families. However, in this section I will describe what I perceive as some of the important aspects of software product families.

From what I know, variation points have emerged as a way to describe the parts where a component can or is allowed to vary. There may be tens of thousands of variation points in a product line – and that might sound frightening. However, in my opinion it is not the number of variation points that should cause most concern with regard to complexity. It is the number of dimensions in which a product family is allowed to vary. Usually that number is in the tens or less, but still this low number generates far more complexity.

The complexity of managing the amount of variability, both in number of places and number of dimensions, can become extremely high. Especially as the product family evolves in time introducing also versions.

4. SCM relations to SVM

Within the SCM community there was an early interest in variants and the handling of variability. Variations that could be confined within a single file and involved only minor pieces of code was studied in [WS88], and a mechanism very similar to conditional compilation was proposed. A later study by [Reichenberger89] distinguished versions and variants as two separate dimensions in which a component could differ. In his approach he put emphasis on a model for the whole product.

So there are already solutions for managing software variability provided by SCM. But are they sufficient for the variability found in today's product families?

5. Position statement

From my point of view it would be interesting to look at SCM from a solutions perspective. Can we "discipline" or stream-line the use of variability in product families such that it can be managed by present SCM techniques – and how?

Equally as interesting would it be to look at software variability management from a problems perspective. To listen to an analysis of what the problems are to discover the limitations of SCM support. This could lead to the development of new SCM techniques to support (part of) these problems.

It is "common wisdom" that evolution and variability should be "kept apart" – but how do you implement that?

More specifically such a discussion could investigate the following five themes:

Top-down vs. bottom-up approach: Is the variation points a top-down approach, whereas current SCM techniques are bottom-up? Are these for large and small variations and which can be supported by SCM and how?

Anticipated vs. unanticipated variability: The latter cannot be designed into the architecture. They could also be considered as sort of proactive and reactive approaches. Can we make a mix – and how should that be done? Would it be possible – and desirable – to use techniques like refactoring to bring unanticipated variability under control?

Life-cycle phases: Is there any difference in the support you need depending on when in the life-cycle you bring in variability? Early (analysis/design), middle (implementation), late (release – or production – and maintenance). Which phases are variation points, SCM- and PDM techniques most suited for?

Evolution vs. variability: These two dimensions should be kept apart. But why – and how?

Traceability: How do we trace and relate a variation point from the analysis to the design to the code to the tests to the documentation?

6. References

[Reichenberger89]: Reichenberger, Christoph: *Orthogonal Version Management*, in proceedings of the 2nd International Workshop on Software Configuration Management, Princeton, New Jersey, October 24-27, 1989.

[WS88]: Winkler, Jürgen F. H. & Stoffel, Clemens: *Program-Variations-in-the-Small*, in proceedings of the 1st International Workshop on Software Version and Configuration Control, Grassau, Germany, January 27-29, 1988.