Variability Issues in Software Product Lines

Jan Bosch[1], Gert Florijn[2], Danny Greefhorst[3],
Juha Kuusela[4], Henk Obbink[5], Klaus Pohl[6]

[1]University of Groningen, Dept. of Computing Science, Groningen, The Netherlands.
Jan.Bosch@cs.rug.nl http://www.cs.rug.nl/~bosch
[2]Software Engineering Research Centre, Utrecht, The Netherlands
florijn@serc.nl http://www.serc.nl
[3]IBM Global Services, Computerweg 8, Amersfoort, The Netherlands
greefhorst@nl.ibm.com http://www.ibm.com/services/nl/
[4]Nokia Research Center, Helsinki, Finland
juha.kuusela@nokia.com http://www.nokia.com
[5]Philips Research
Henk.Obbink@philips.com http:
[6]University of Essen, Software Systems Engineering, Essen, Germany
pohl@informatik.uni-essen.de http://www.sse.uni-essen.de

Abstract. Software product lines (or system families) have achieved considerable adoption by the software industry. A software product line captures the commonalities between a set of products while providing for the differences. Differences are managed by delaying design decisions, thereby introducing variation points. The whole of variation points is typically referred to as the variability of the software product line. Variability management is, however, not a trivial activity and several issues exist, both in general as well as specific to individual phases in the lifecycle. This paper identifies and describes several variability issues based on practical experiences and theoretical understanding of the problem domain.

1 Introduction

Software product lines (or system families) provide a highly successful approach to strategic reuse of assets within an organization. A standard software product line consists of a product line architecture, a set of software components and a set of products. A product consists of a product architecture, derived from the product line architecture, a set of selected and configured product line components and product specific code.

Software product lines come in many different forms. In some cases, the architecture of the product line is used by all products without being adapted, whereas in other cases the product architectures may deviate substantially. Similarly, in certain cases, there is exactly one, configurable, component implementation associated with each architectural component, whereas in other product lines, multiple component implementations exist for an architectural component.

The different forms discussed above exploit different variability mechanisms for describing the differences between the products. In our discussion, we will consider variability in the space dimension, e.g. a component is used in multiple products but needs to exhibit different behaviours, and the time dimension, i.e. a software artefact evolves over time. As we will see, the same mechanisms are used for achieving both dimensions of variability.

In software product lines, variability is made explicit through *variation points.* A variation point represents a delayed design decision. When the architect or designer decides to delay the design decision, he or she has to design a variation point. The design of the variation point requires a number of steps, i.e. the separation of the stable and variant behaviour, the definition of an interface between these types of behaviour, the design of a variant management mechanism and the implementation of one or more variants. In the lifecycle stages before the design of the variation point, we consider the variation point to be implicit. At or after the stage at which the variation point is designed, the variation point is explicit. An explicit variation point can be bound to a particular variant. For each variation point, the set of variants may be open, i.e. more variants can be added, or closed, i.e. no more variants can be added. Within the existing set, different variants can be bound. Finally, a variation point can be permanently bound, i.e. one variant has been bound permanently.

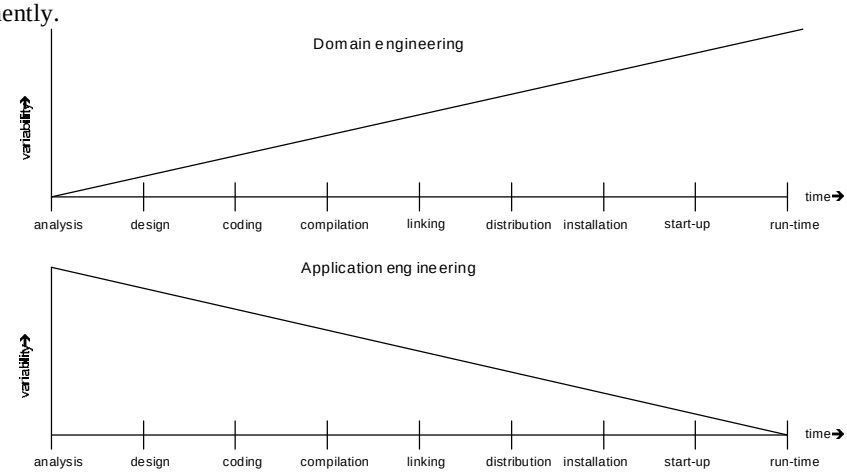

Fig. 1. Managing variability in time

Software development in software product lines can be viewed as being organized in two stages, i.e. domain engineering and application engineering. Domain engineering is, among others, concerned with identifying the commonality and variability for the products in the product line and implementing the shared artefacts such that the commonality can be exploited while preserving the required variability. During application engineering individual products are developed by selecting and configuring shared artefacts and, where necessary, adding product-specific

uring shared artefacts and, where necessary, adding product-specific extensions. The variability of the software artefacts evolves according to Figure 1, i.e. during domain engineering new variation points are introduced, whereas during application engineering these variation points are bound to selected variants.

The aim and contribution of this article is to identify and discuss the core issues of variability management. These issues were identified during a discussion group meeting at the ESAPS[1] derivation workshop in Bad Kohlgrub held in April 2001. We believe that by increasing the awareness of these issues, practitioners are better able to avoid some of the difficulties associated with software product lines whereas these issues present a research agenda to the academic and industrial researchers.

The remainder of this paper is organized as follows. In the next section, we describe a number of general issues. Section 3 discusses the issues associated with the identification and design of variability during domain engineering whereas section 4 discusses the resolution of variation points during application engineering, especially concerning run-time. Section 5 discusses the evolution of variability. Related work is discussed in section 6 and we conclude in section 7.

2. General Issues

The variability in a software product line is a concern in all phases of the life cycle. Most variability management issues are specific for each life cycle phase,. In this section we discuss the general issues.

- **First-class representation**: Most of the modelling mechanisms for variability do not have a first-class representation for features and variation points. As a result, it is difficult to see the variability at the requirements and realisation level. In particular, it is difficult to assess the impact of changes. There is a need for a standard process, notation and exchange format for describing variability within different modelling mechanisms.

- **Implicit dependencies**: Dependencies between architectural elements and features are seldom made explicit. As a result it is often unclear what parts of the product line architecture are needed for a specific product.
- **Tool support**: The number of variation points and associated variants as well as the dependencies may be so high that the effort required for managing these manually becomes prohibitive. Consequently, tool support for automatically maintaining variability information is critical. Current software configuration management tools were developed with the aim to support versioning rather than to support variability and, consequently, fail to support important features, in particular related to variability management in phases different from compilation and linking.
- **Phase selection**: As we discussed in the introduction, variation points are introduced, extended and bound at particular phases in the lifecycle. Selection of the optimal phase for each of the aforementioned activities has a considerable impact on the flexibility of the system as well as the development and maintenance cost. However, we lack methods, techniques, guidelines and heuristics for trade offs between different alternatives.
- **Mechanism selection**: Variability mechanisms are often chosen without consideration for specific advantages and disadvantages of specific mechanisms. Discovering late during development that the wrong mechanism was chosen, e.g. because

---

[1] ESAPS (Engineering Software Architectures, Processes and Platforms for System-Families) is a European Eureka/ITEA project. For more information, see www.esi.es/esaps/esaps.html.