

Preface

There are several books on product line engineering, but most of these books either introduce specific product line techniques or include brief summaries of industrial cases or researches. From these sources, it is difficult to gain a comprehensive understanding of the various dimensions and aspects of software variability that product line engineering practitioners and researchers must understand. The book aims to address this gap by providing a comprehensive reference on the notion of variability modeling in the context of software product line engineering, to give an overview of the techniques proposed for variability modeling, and to give a general perspective on software variability management. We believe that practitioners as well as researchers and computer science students will gain a new insight into software, software engineering, and variability in product line engineering.

The most important attribute of software is the “softness” of software, i.e., software that is easy (cost-effective) to modify and adapt to evolving requirements or changing operating environments, easy to port on different hardware or software platforms, and easy to reuse for development of similar applications. The “softness” of software cannot be attained without engineering it into software. In order to “embed” softness into software, we need to understand the “space” of the commonality and variability of a family of related systems (i.e., a product line) and its evolution, and then organize and codify the knowledge gathered as a commonality and variability model. With this understanding, we can engineer software applying various design principles and embedding variation points that can later be bound with variants.

Once the initial variability of a software product line has been established and implemented, the focus shifts towards evolution of the provided variability in response to changes in the variability required from the software product line. Over time, new products are added to a product line, old products are removed, and the functionality that used to be highly differentiating and only used in the high-end products of the product line commodities is included in all products, removing the need for variation points. Thus, during evolution, the focus of variability management is as much on removing unnecessary variability as it is on adding new variation points in response to new needs. The focus on removing variability is important as the complexity of hundreds or thousands of variation points can easily become unwieldy.

One aspect contributing to the complexity of software variability management is the dependencies between variation points and between the variants available at each variation point. Where industrial software product lines frequently hold well over 1,000 variation points, the total number of variants is even larger. The variants cannot be freely selected independent of all other selected variants, but instead there are dependencies that need to be respected. This leads to a situation where the number of variation points is over a thousand, the number of variants a multiple of that, and the number of dependencies between variants and between variation points is of a similar or larger number. This explains the importance of intentional management of software variability: even though the inherent complexity of variability is already quite high, the total complexity easily becomes unmanageable if not kept under control.

A final factor increasing the importance of software variability is concerned with later binding of software variation points. Traditional pre-deployment configuration of products allows for testing of the specific configuration as a safety net for avoiding inconsistent configurations. However, run-time dynamicity is increasing in importance for virtually all software-intensive systems. In those situations, testing of the resulting configuration after a run-time change is complicated and often only the most basic of system integrity is verified. The trend towards late binding is indicative of the importance of software variability management even outside the traditional area of software product lines and is now becoming important for large software products that have significant installation time, startup time, and run-time configuration taking place. Consequently, proper management of variability avoiding inconsistent configurations at run-time through the use of first-class models is particularly important.

Above, we have raised four reasons to stress the importance of software variability management, i.e., modeling multiple products in a product line, evolution, complexity, and the shift towards later, even run-time, binding of variation points. As is illustrated in the industrial experiences part of the book, there are no theoretical problems without any bearing on industrial practice, but rather challenges originating in industrial practice that the software engineering research community has responded to. In our experience, mature software product lines may have ten thousand variation points or more and the number of legal configurations of products in the product line may number in the millions. Also, in our writing of the book and the interaction with contributing authors, we are increasingly becoming convinced that software variability management is evolving into a field of its own, rather than a subfield of software product lines. In all systems where configuration and run-time dynamism are important, software variability management offers a powerful toolbox to deal with the resulting complexity, independent of the system being part of a software product line or not.

From a software engineering research perspective, software variability management represents a complex, multifaceted problem that intersects with several traditional topics, including, among several others, software configuration management, run-time dynamism, domain specific languages, modeling, and software architecture. The field has borrowed techniques from these traditional fields, but in return also contributes back with new insights, approaches, and techniques.

The book is organized in four main parts which guide the reader into the various aspects and dimensions on software variability. Each chapter briefly summarizes “*What you will learn in this chapter*”; so expert and non-expert? Readers can easily locate what topics they will find, but it also describes areas of practice for the applicability of the concepts explained.

In Part I, we intentionally drive the reader to the major topics on software variability modeling, but as we do not have a specific chapter for variability management, the chapters included in this part should be seen as different sides of the management perspective. First, we introduce the paradigm of software product line engineering in Chap. 1, where Product Line Engineering is compared with traditional Software Engineering and the role of software variability management is highlighted for the current practice of product lines. We then explore various dimensions of commonality and variability (C&V) in Chaps. 2 and 3, separating C&V modeling into problem and solution space modeling, and constraints specification. Managing traceability between various C&V models and the notion of variability in time and space is also discussed. The dimension of feature binding time, the implications of deciding a specific binding time, and its importance for the software development life cycle are discussed in Chap. 4, which also provides a renewed taxonomy of different binding times. Chapters 5 and 6 describe ways to implement and configure software variability. In Chap. 5 we outline from a high-level perspective various mechanisms for implementing software variability, and how variability implementation mechanisms affect the architecture, components, and code levels. We did not go into the specific implementation details as many of the mechanisms described in the chapter depend on the programming language selected. Once product line variability is embedded into the product line asset (code) using various mechanisms, we should be able to configure products from the asset. Chap. 6 focuses on processes of product derivation activities for pre- and post-deployment times, with special mention of the configuration tasks of software products at run-time and reconfiguration activities. Because of the complexity of C&V models and complex interrelationships among them, visualizing the relationships between modeling elements is useful and enhances understandability and maintainability. Techniques for visualization are discussed in Chap. 7. Finally, we conclude this part of the book in Chap. 8 with a description on how different life-cycle products are related to each other in terms of variability when feature models are considered a first-class artifact for any product line engineering process.

Part II of the book describes an overview of research and commercial tools, from Chaps. 9–12. Three research tools, COVAMOF, PLUM, and FaMa, address different aspects of variability management as they provide automatic support for managing, configuring, and testing feature models with other related software artifacts. The commercial tool `pure::variants` is a variability management suite that evolves from the original FODA feature model to support the problem and solution spaces for describing variant configurations.

Part III shows the most practical viewpoint of the book as we collect three different industry cases on how variability is managed in real industry projects. Chapter 13 provides the view of Philips Healthcare Systems where product line engineering is used extensively to manage the complexity and the diversity of the Philips systems that rely on C&V and configuration properties of the Philips Software Product Line (SPL). In Chap. 14 Toshiba researchers use a product line strategy to describe the variability in power plants software for managing an automatic control system where complex rules model the relationships between events, conditions, and actions. Chapter 15 from BigLever Software focuses on Second Generation Product Line Engineering (2GPLE) activities and tools and applied to an industrial case at General Motors. In this chapter we can discover the differences between traditional SPLE activities (first generation) and those suggested for a second generation (2GPLE), where variability is described and managed consistently and traceable across the full engineering life cycle and configuration management is simplified.

Part IV concludes the book and encompasses six different chapters focused on emerging topics about software variability that, currently, are under research. The diversity of topics include dynamic software product lines, variability in autonomic computing and web services, the relationship and role of variability in service-oriented product lines, the impact and use of design decisions in conjunction with variability models, and finally, how variability is realized using aspect orientation. We believe that there are more interesting research topics that can be discussed with more detail, but this part of the book provides and suggests the readers current and future trends where variability can be applied to manage the diversity of products in different types of systems or how other software engineering techniques can be also applied with variability models and vice versa.

As authors and editors, we feel that the book presents an important contribution both to the industrial practice of software product lines and software engineering more broadly and to the software engineering research community. We have strived to capture the current state of the art and state of practice in the chapters and to indicate important, open research challenges as well as pitfalls for industrial practitioners to be aware of. We hope that the book can serve as a platform for the community of researchers and practitioners in software variability management, allowing the community to develop the next set of solutions, techniques, and methods to address this complicated and yet fascinating field in software engineering.

Madrid, Spain
Gothenburg, Sweden
Pohang, Republic of Korea
January 2013

Rafael Capilla
Jan Bosch
Kyo-Chul Kang

Systems and Software Variability Management

Concepts, Tools and Experiences

Capilla, R.; Bosch, J.; Kang, K.-C. (Eds.)

2013, XIV, 317 p., Hardcover

ISBN: 978-3-642-36582-9