Günter Böckle
Klaus Pohl
Frank van der Linden

2

A Framework for Software Product Line Engineering

In this chapter you will learn:

- *The principles of software product line engineering subsumed by our software product line engineering framework.*
- *The difference between domain engineering and application engineering, which are the two key processes of software product line engineering.*
- *Where variability of the product line is defined and where it is exploited.*
- *The structure of this book, which is derived from the framework.*

2.1 Introduction

Our framework for software product line engineering incorporates the central concepts of traditional product line engineering, namely the use of platforms and the ability to provide mass customisation.

Platform artefacts

A platform is, in the software context, a collection of reusable artefacts (Definition 1-4). These artefacts have to be reused in a consistent and systematic way in order to build applications. Reusable artefacts encompass all types of software development artefacts such as requirements models, architectural models, software components, test plans, and test designs.

Planning for reuse

The experience from reuse projects in the 1990s shows that without proper planning the costs for reuse may be higher than for developing the artefacts from scratch. It is therefore crucial to plan beforehand the products for which reuse is sensible, together with the features that characterise these products. The planning for reuse continues throughout the whole development process.

Mass customisation through variability

To facilitate mass customisation (Definition 1-1) the platform must provide the means to satisfy different stakeholder requirements. For this purpose the concept of variability is introduced in the platform. As a consequence of applying this concept, the artefacts that can differ in the applications of the product line are modelled using variability.

The following sections outline our software product line engineering framework.

2.2 Two Development Processes

The software product line engineering paradigm separates two processes:

The software product line engineering paradigm separates two processes (see e.g. [Weiss and Lai 1999; Boeckle et al. 2004b; Pohl et al. 2001b, V.d. Linden 2002]):

Establishing the platform

- *Domain engineering*: This process is responsible for establishing the reusable platform and thus for defining the commonality and the variability of the product line (Definition 2-1). The platform consists of all types of software artefacts (requirements, design, realisation, tests, etc.). Traceability links between these artefacts facilitate systematic and consistent reuse.

Deriving applications

- *Application engineering*: This process is responsible for deriving product line applications from the platform established in domain engineering; see Definition 2-2. It exploits the variability of the product line and

ensures the correct binding of the variability according to the applications' specific needs.

Separation of concerns

The advantage of this split is that there is a separation of the two concerns, to build a robust platform and to build customer-specific applications in a short time. To be effective, the two processes must interact in a manner that is beneficial to both. For example, the platform must be designed in such a way that it is of use for application development, and application development must be aided in using the platform.

Flexibility and variability

The separation into two processes also indicates a separation of concerns with respect to variability. Domain engineering is responsible for ensuring that the available variability is appropriate for producing the applications. This involves common mechanisms for deriving a specific application. The platform is defined with the right amount of flexibility in many reusable artefacts. A large part of application engineering consists of reusing the platform and binding the variability as required for the different applications.

> ***Definition 2-1:*** *Domain Engineering*
>
> Domain engineering is the process of software product line engineering in which the commonality and the variability of the product line are defined and realised.

> ***Definition 2-2:*** *Application Engineering*
>
> Application engineering is the process of software product line engineering in which the applications of the product line are built by reusing domain artefacts and exploiting the product line variability.

2.3 Overview of the Framework

ITEA projects

Our software product line engineering framework has its roots in the ITEA projects ESAPS, CAFÉ, and FAMILIES [V.d. Linden 2002; Boeckle et al. 2004b; CAFÉ 2004] and is based on the differentiation between the domain and application engineering processes proposed by Weiss and Lai [Weiss and Lai 1999]. The framework is depicted in Fig. 2-1.

Domain engineering

The *domain engineering process* (depicted in the upper part of Fig. 2-1) is composed of five key sub-processes: product management, domain requirements engineering, domain design, domain realisation, and domain testing. The domain engineering process produces the platform including the commonality of the applications and the variability to support mass customi-

sation. We briefly describe the domain engineering process and its sub-processes in Section 2.4.

Application engineering

The *application engineering process* (depicted in the lower part of Fig. 2-1) is composed of the sub-processes application requirements engineering, application design, application realisation, and application testing. We briefly describe the application engineering process and its sub-processes in Section 2.6.

Domain and application artefacts

The framework differentiates between different kinds of development artefacts (Definition 2-3): domain artefacts and applications artefacts. The *domain artefacts* (Definition 2-4) subsume the platform of the software product line. We briefly characterise the various artefacts in Section 2.5. The *application artefacts* (Definition 2-5) represent all kinds of development artefacts of specific applications. We briefly characterise these artefacts in Section 2.7. As the platform is used to derive more than one application, application engineering has to maintain the application-specific artefacts for each application separately. This is indicated in the lower part of Fig. 2-1.

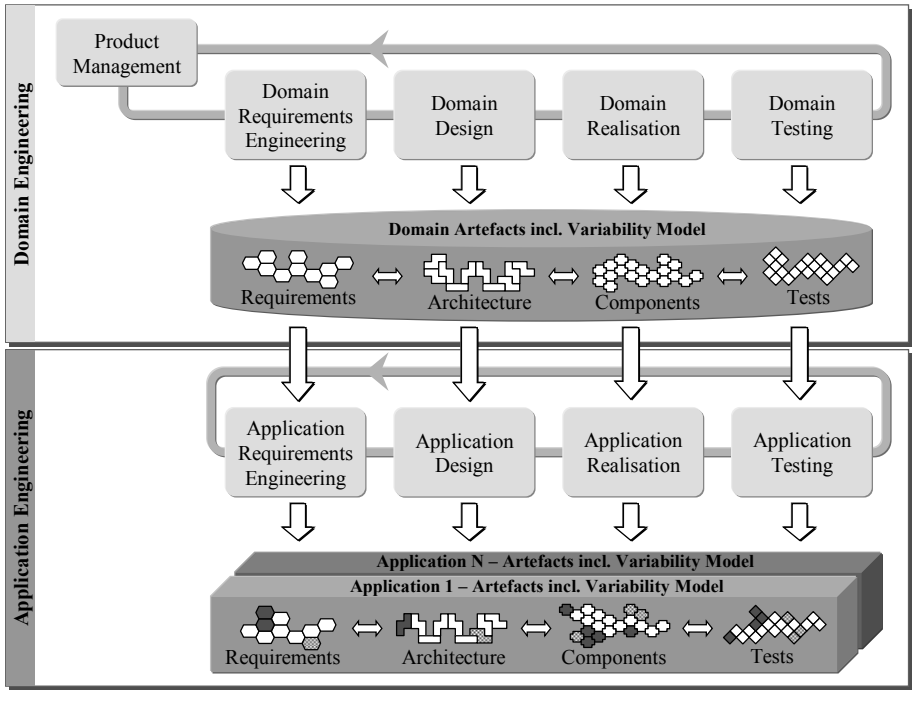

Fig. 2-1: The software product line engineering framework

No sequential order implied

Note that neither the sub-processes of the domain and application engineering processes, nor their activities, have to be performed in a sequential order. We have indicated this by a loop with an arrow in Fig. 2-1 for each process.

Combination with existing processes

In this book, we define the key activities that have to be part of each product line engineering process. The order in which they are performed depends on the particular process that is established in an organisation. Thus, the sub-processes and their activities described in this book can be combined with existing development methods such as RUP (Rational Unified Process, see [Kruchten 2000]), the spiral model [Boehm 1988], or other development processes.

Organisation-specific adaptation

When the domain engineering process and the application engineering process are embedded into other processes of an organisation, each sub-process depicted in Fig. 2-1 gets an organisation-specific internal structure. Nevertheless, the activities presented in this book have to be present. An example of an organisation-specific process is the FAST process presented in [Weiss and Lai 1999].

> ***Definition 2-3:*** *Development Artefact*
>
> A development artefact is the output of a sub-process of domain or application engineering. Development artefacts encompass requirements, architecture, components, and tests.

> ***Definition 2-4:*** *Domain Artefacts*
>
> Domain artefacts are reusable development artefacts created in the sub-processes of domain engineering.

> ***Definition 2-5:*** *Application Artefacts*
>
> Application artefacts are the development artefacts of specific product line applications.

2.4 Domain Engineering

Main goals

The key goals of the domain engineering process are to:

- Define the commonality and the variability of the software product line.
- Define the set of applications the software product line is planned for, i.e. define the scope of the software product line.

- Define and construct reusable artefacts that accomplish the desired variability.

The goals of domain engineering are accomplished by the domain engineering sub-process. Each of them has to:

- Detail and refine the variability determined by the preceding sub-process.
- Provide feedback about the feasibility of realising the required variability to the preceding sub-process.

Five sub-processes

The domain engineering part of the software product line engineering framework is highlighted in Fig. 2-2. We briefly explain the domain engineering sub-processes in this section, whereas domain artefacts are explained separately in Section 2.5.

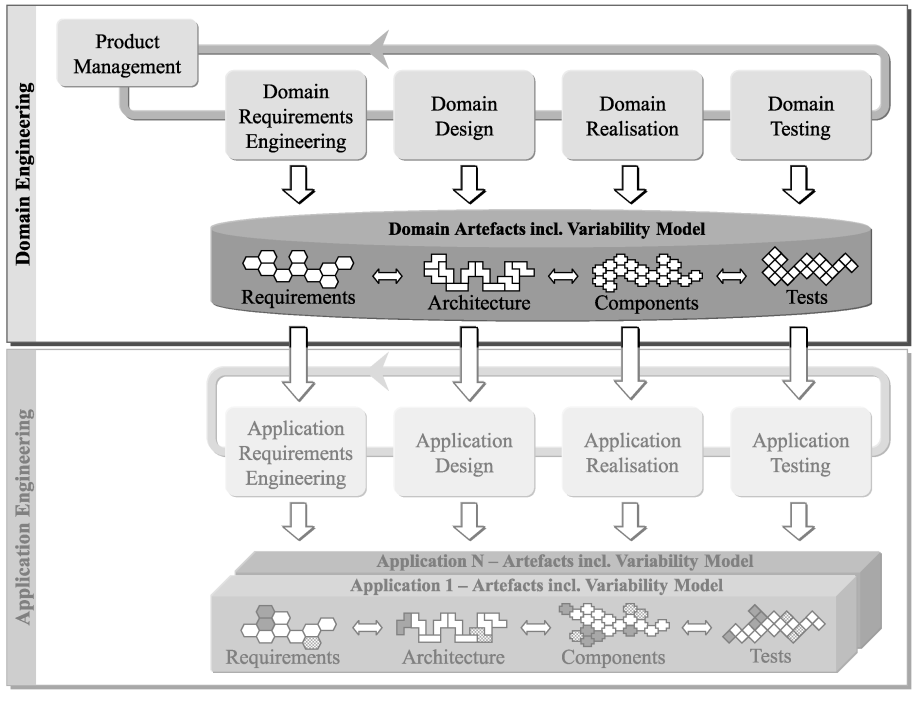

Fig. 2-2: The domain engineering process

2.4.1 Product Management

Scope of the product line

Product management deals with the economic aspects of the software product line and in particular with the market strategy. Its main concern is the management of the product portfolio of the company or business unit. In

product line engineering, product management employs scoping techniques to define what is within the scope of the product line and what is outside.

Input and output

The *input* for product management consists of the company goals defined by top management. The *output* of product management is a product roadmap that determines the major common and variable features[9] of future products as well as a schedule with their planned release dates. In addition, product management provides a list of existing products and/or development artefacts that can be reused for establishing the platform.

Differences from single-system engineering

Product management for software product lines differs from product management for single systems for the following reasons:

- The platform has an essential strategic meaning for the company. The introduction and elimination of an entire platform have a strong influence on entrepreneurial success.
- A major strength of software product line engineering is the generation of a multitude of product variants at reasonable cost.
- The products in the product portfolio are closely related as they are based on a common platform.
- Product management anticipates prospective changes in features, legal constraints, and standards for the future applications of the software product line and formulates (models) the features accordingly. This means that the evolution of market needs, of technology, and of constraints for future applications is taken into account.

We deal with the principles of product management, activities, and artefacts in Chapter 9.

2.4.2 Domain Requirements Engineering

Elicitation and documentation

The domain requirements engineering sub-process encompasses all activities for eliciting and documenting the common and variable requirements of the product line.

Input and output

The *input* for this sub-process consists of the product roadmap. The *output* comprises reusable, textual and model-based requirements and, in particular, the variability model of the product line. Hence, the output does not include the requirements specification of a particular application, but the common and variable requirements for all foreseeable applications of the product line.

[9] A feature is an abstract requirement (see Definition 5-4 for the definition of "feature" by Kang et al.).

Differences from single-system engineering

Domain requirements engineering differs from requirements engineering for single systems because:

- The requirements are analysed to identify those that are common to all applications and those that are specific for particular applications (i.e. that differ among several applications).
- The possible choices with regard to requirements are explicitly documented in the variability model, which is an abstraction of the variability of the domain requirements.
- Based on the input from product management, domain requirements engineering anticipates prospective changes in requirements, such as laws, standards, technology changes, and market needs for future applications.

The artefacts and activities of domain requirements engineering are described in detail in Chapter 5 and Chapter 10.

2.4.3 Domain Design

Definition of reference architecture

The domain design sub-process encompasses all activities for defining the reference architecture of the product line. The reference architecture provides a common, high-level structure for all product line applications.

Input and output

The *input* for this sub-process consists of the domain requirements and the variability model from domain requirements engineering. The *output* encompasses the reference architecture and a refined variability model that includes so-called internal variability (e.g. variability that is necessary for technical reasons).

Differences from single-system engineering

Domain design differs from design for single systems because:

- Domain design incorporates configuration mechanisms into the reference architecture to support the variability of the product line.
- Domain design considers flexibility from the very first, so that the reference architecture can be adapted to the requirements of future applications.
- Domain design defines common rules for the development of specific applications based on the reference architecture.
- Domain design designates reusable parts, which are developed and tested in domain engineering, as well as application-specific parts, which are developed and tested in application engineering.

The artefacts and activities of domain design are described in detail in Chapter 6 and Chapter 11.

2.4.4 Domain Realisation

Detailed design and implementation

The domain realisation sub-process deals with the detailed design and the implementation of reusable software components.

Input and output

The *input* for this sub-process consists of the reference architecture including a list of reusable software artefacts to be developed in domain realisation. The *output* of domain realisation encompasses the detailed design and implementation assets of reusable software components.

Differences from single-system engineering

Domain realisation differs from the realisation of single systems because:

- The result of domain realisation consists of loosely coupled, configurable components, not of a running application.
- Each component is planned, designed, and implemented for the reuse in different contexts, i.e. the applications of the product line. The interface of a reusable component has to support the different contexts.
- Domain realisation incorporates configuration mechanisms into the components (as defined by the reference architecture) to realise the variability of the software product line.

The artefacts and activities of domain realisation are described in detail in Chapter 7 and Chapter 12.

2.4.5 Domain Testing

Validation of reusable components

Domain testing is responsible for the validation and verification of reusable components. Domain testing tests the components against their specification, i.e. requirements, architecture, and design artefacts. In addition, domain testing develops reusable test artefacts to reduce the effort for application testing.

Input and output

The *input* for domain testing comprises domain requirements, the reference architecture, component and interface designs, and the implemented reusable components. The *output* encompasses the test results of the tests performed in domain testing as well as reusable test artefacts.

Differences from single-system engineering

Domain testing differs from testing in single-system engineering, because:

- There is no running application to be tested in domain testing. Indeed, product management defines such applications, but the applications are available only in application testing. At first glance, only single components and integrated chunks composed of common parts can be tested in domain testing.
- Domain testing can embark on different strategies with regard to testing integrated chunks that contain variable parts. It is possible to create a

sample application, to predefine variable test artefacts and apply them in application testing, or to apply a mixture of the former two strategies.

We describe the artefacts produced by domain testing in Chapter 8 and deal with product line test strategies and techniques in Chapter 13.

2.4.6 Other Software Quality Assurance Techniques

Inspections, reviews, and walkthroughs

Besides testing, other software quality assurance techniques are also applicable to software product line engineering, most notably inspections, reviews, and walkthroughs. These techniques have to be integrated into the domain and application engineering processes.

Techniques for single systems

To our knowledge, specialised techniques for software product line inspections, reviews, or walkthroughs have not been proposed. There is also a lack of experience reports identifying required adaptations of inspection, review, and walkthrough techniques known from the development of single software systems. We thus refer the interested reader to the standard literature on software inspections, reviews, and walkthroughs such as [Fagan 1976; Fagan 1986; Freedman and Weinberg 1990; Gilb and Graham 1993; Yourdon 1989].

2.5 Domain Artefacts

Common platform

Domain artefacts (or domain assets; see Definition 2-4) compose the platform of the software product line and are stored in a common repository. They are produced by the sub-processes described in Section 2.4. The artefacts are interrelated by traceability links to ensure the consistent definition of the commonality and the variability of the software product line throughout all artefacts. In the following, we briefly characterise each kind of artefact including the variability model.

2.5.1 Product Roadmap

Major features of all applications

The product roadmap describes the features of all applications of the software product line and categorises the feature into common features that are part of each application and variable features that are only part of some applications. In addition, the roadmap defines a schedule for market introduction. The product roadmap is a plan for the future development of the product portfolio. Its role in domain engineering is to outline the scope of the platform and to sketch the required variability of the product line. Its role in application engineering is to capture the feature mix of each planned application.

Product roadmap not in framework picture

Note that the output of product management (the product roadmap) is not contained in the framework picture. The main reason for this is that the product roadmap is not a software development artefact in the common sense. Moreover, it guides both domain and application engineering, and is not an artefact to be reused in application engineering. In domain engineering, it guides the definition of the commonality and the variability of the software product line. In application engineering it guides the development of the specific products. We thus decided to define the product roadmap neither as a domain nor as an application artefact. We deal with the essential techniques for defining the product roadmap in Chapter 9.

2.5.2 Domain Variability Model

Product line variability

The domain variability model defines the variability of the software product line. It defines what can vary, i.e. it introduces variation points for the product line. It also defines the types of variation offered for a particular variation point, i.e. it defines the variants offered by the product line. Moreover, the domain variability model defines variability dependencies and variability constraints which have to be considered when deriving product line applications. Last but not least, the domain variability model interrelates the variability that exists in the various development artefacts such as variability in requirements artefacts, variability in design artefacts, variability in components, and variability in test artefacts. It thus supports the consistent definition of variability in all domain artefacts.

Orthogonal variability model

We describe the variability model in greater detail in Chapter 4. To distinguish our variability model from the definition of variability within other development artefacts, we call it the “orthogonal variability model”.

2.5.3 Domain Requirements

Reusable requirements artefacts

Domain requirements encompass requirements that are common to all applications of the software product line as well as variable requirements that enable the derivation of customised requirements for different applications. Requirements are documented in natural language (textual requirements) or by conceptual models (model-based requirements). Variability occurs in functional as well as in quality requirements. In Chapter 5, we elaborate on modelling variability in requirements using the orthogonal variability model.

2.5.4 Domain Architecture

Core structure and common texture

The domain architecture or *reference architecture* determines the structure and the texture of the applications in the software product line. The structure determines the static and dynamic decomposition that is valid for all appli-

cations of the product line. The texture is the collection of common rules guiding the design and realisation of the parts, and how they are combined to form applications. Variability in the architecture is documented by refining the orthogonal variability model and adding internal variability (i.e. variability that is only visible to the engineers). The architectural texture defines common ways to deal with variability in domain realisation as well as in application design and application realisation. Chapter 6 elaborates on the documentation of variability in design artefacts using the orthogonal variability model.

2.5.5 Domain Realisation Artefacts

Detailed design and implementation

Domain realisation artefacts comprise the design and implementation artefacts of reusable software components and interfaces. The design artefacts encompass different kinds of models that capture the static and the dynamic structure of each component. The implementation artefacts include source code files, configuration files, and makefiles. Components realise variability by providing suitable configuration parameters in their interface. In addition to being configurable, each component may exist in different variants to realise large differences in functionality and/or quality. We elaborate on variability in domain realisation artefacts in Chapter 7.

2.5.6 Domain Test Artefacts

Reusable test designs

Domain test artefacts include the domain test plan, the domain test cases, and the domain test case scenarios. The domain test plan defines the test strategy for domain testing, the test artefacts to be created, and the test cases to be executed. It also defines the schedule and the allocation of resources for domain test activities. The test cases and test case scenarios provide detailed instructions for the test engineer who performs a test and thus make testing traceable and repeatable. We include variability definitions in domain test artefacts to enable the large-scale reuse of test artefacts in application testing. We deal with the documentation of variability in test artefacts in Chapter 8.

2.6 Application Engineering

Main goals

The key goals of the application engineering process are to:

- Achieve an as high as possible reuse of the domain assets when defining and developing a product line application.
- Exploit the commonality and the variability of the software product line during the development of a product line application.

- Document the application artefacts, i.e. application requirements, architecture, components, and tests, and relate them to the domain artefacts.
- Bind the variability according to the application needs from requirements over architecture, to components, and test cases.
- Estimate the impacts of the differences between application and domain requirements on architecture, components, and tests.

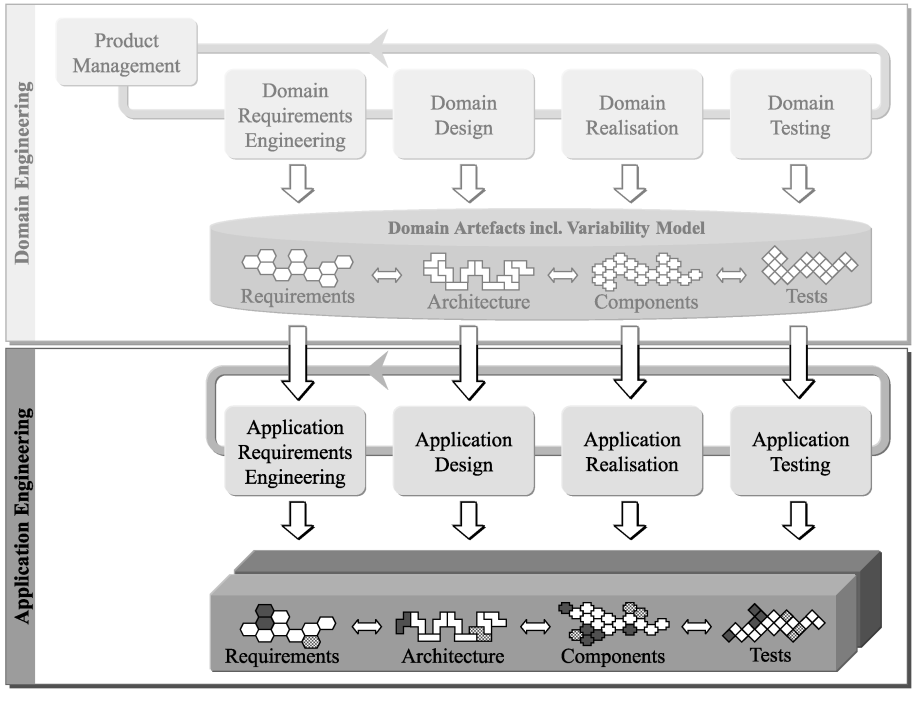

Fig. 2-3: Application engineering

Four sub-processes

The framework introduces four application engineering sub-processes: application requirements engineering, application design, application realisation, and application test. Each of the sub-processes uses domain artefacts and produces application artefacts. Figure 2-3 highlights the application engineering part of the software product line engineering framework. We characterise the application engineering sub-processes in this section. Application artefacts are described in Section 2.7.

2.6.1 Application Requirements Engineering

Specification of applications

The application requirements engineering sub-process encompasses all activities for developing the application requirements specification. The achievable amount of domain artefact reuse depends heavily on the applica-

tion requirements. Hence, a major concern of application requirements engineering is the detection of deltas between application requirements and the available capabilities of the platform.

Input and output

The *input* to this sub-process comprises the domain requirements and the product roadmap with the major features of the corresponding application. Additionally, there may be specific requirements (e.g. from a customer) for the particular application that have not been captured during domain requirements engineering. The *output* is the requirements specification for the particular application.

Differences from single-system engineering

Application requirements engineering differs from requirements engineering for single systems for the following reasons:

- Requirements elicitation is based on the *communication of the available commonality and variability* of the software product line. Most of the requirements are not elicited anew, but are derived from the domain requirements.
- During elicitation, *deltas* between application requirements and domain requirements must be detected, evaluated with regard to the required adaptation effort, and documented suitably. If the required adaptation effort is known early, *trade-off decisions* concerning the application requirements are possible to reduce the effort and to increase the amount of domain artefact reuse.

We deal with the specific activities of application requirements engineering in Chapter 15.

2.6.2 Application Design

Specialisation of reference architecture

The application design sub-process encompasses the activities for producing the application architecture. Application design uses the reference architecture to instantiate the application architecture. It selects and configures the required parts of the reference architecture and incorporates application-specific adaptations. The variability bound in application design relates to the overall structure of the considered system (e.g. the specific hardware devices available in the system).

Input and output

The *input* for application design consists of the reference architecture and the application requirements specification. The *output* comprises the application architecture for the application at hand.

Differences from single-system engineering

Application design differs from the design process for single systems for the following reasons:

- Application design does not develop the application architecture from scratch, but derives it from the reference architecture by binding vari-

ability, i.e. making specific choices at places where the reference architecture offers different variants to choose from.

- Application design has to comply with the rules defined in the texture of the reference architecture. The rules cover the binding of variability as well as the incorporation of application-specific adaptations.
- Application design must evaluate the realisation effort for each required adaptation and may reject structural changes that would require a similar effort as for developing the application from scratch.

We elaborate on the key problems and solutions of application design in Chapter 16.

2.6.3 Application Realisation

Component configuration and development

The application realisation sub-process creates the considered application. The main concerns are the selection and configuration of reusable software components as well as the realisation of application-specific assets. Reusable and application-specific assets are assembled to form the application.

Input and output

The *input* consists of the application architecture and the reusable realisation artefacts from the platform. The *output* consists of a running application together with the detailed design artefacts.

Differences from single-system engineering

Application realisation differs from the realisation of single systems because:

- Many components, interfaces, and other software assets are not created anew. Instead, they are derived from the platform by binding variability. Variability is bound, e.g. by providing specific values for component-internal configuration parameters.
- Application-specific realisations must fit into the overall structure, e.g. they must conform to the reusable interfaces. Many detailed design options are predetermined by the architectural texture. Application-specific components can often be realised as variants of existing components that are already contained in the platform.

We deal with the challenges of application realisation in Chapter 17.

2.6.4 Application Testing

Complete application test

The application testing sub-process comprises the activities necessary to validate and verify an application against its specification.

Input and output

The *input* for application testing comprises all kinds of application artefacts to be used as a test reference,[10] the implemented application, and the reusable test artefacts provided by domain testing. The *output* comprises a test report with the results of all tests that have been performed. Additionally, the detected defects are documented in more detail in problem reports.

Differences from single-system engineering

The major differences from single-system testing are:

- Many test artefacts are not created anew, but are derived from the platform. Where necessary, variability is bound by selecting the appropriate variants.
- Application testing performs additional tests in order to detect defective configurations and to ensure that exactly the specified variants have been bound.
- To determine the achieved test coverage, application testing must take into account the reused common and variable parts of the application as well as newly developed application-specific parts.

We elaborate on the specific challenges and the activities of application testing in Chapter 18.

2.7 Application Artefacts

Traceability between application artefacts

Application artefacts (or application assets) comprise all development artefacts of a specific application including the configured and tested application itself. They are produced by the sub-processes described in Section 2.6. The application artefacts are interrelated by traceability links. The links between different application artefacts are required, for instance, to ensure the correct binding of variability throughout all application artefacts.

Traceability between domain and application

Many application artefacts are specific instances of reusable domain artefacts. The orthogonal variability model is used to bind variability in domain artefacts consistently in the entire application. Traceability links between application artefacts and the underlying domain artefacts are captured to support the various activities of the application engineering sub-processes. These links also support the consistent evolution of the product line. For example, if a domain artefact changes, the application artefacts affected by this change can be easily determined. In the following, we briefly characterise each kind of application artefact.

[10] The artefacts used as a test reference comprise the application requirements specification, the application architecture, and the component and interface designs.

2.7.1 Application Variability Model

Variability bindings for applications

The application variability model documents, for a particular application, the binding of the variability, together with the rationales for selecting those bindings. It is restricted by the variability dependencies and constraints defined in the domain variability model. Moreover, the application variability model documents extensions to the domain variability model that have been made for the application. For example, it documents if a new variant has been introduced for the application. It also documents if existing variants have been adapted to match the application requirements better and if new variation points have been introduced for the application. Briefly, the application variability model documents the variability bindings made and all extensions and changes made for a particular application. Note that, similar to other application artefacts, a separate application variability model is introduced for each product line application. We deal with the definition of the application variability model in Chapter 15.

2.7.2 Application Requirements

Complete specification

Application requirements constitute the complete requirements specification of a particular application. They comprise reused requirements as well as application-specific requirements. The reuse of domain requirements involves the use of the orthogonal variability model to bind the available variability. Application-specific requirements are either newly developed requirements or reused requirements that have been adapted. Chapter 15 elaborates on how to define the application requirements specification.

2.7.3 Application Architecture

Specific instance of reference architecture

The application architecture determines the overall structure of the considered application. It is a specific instance of the reference architecture. For the success of a product line, it is essential to reuse the reference architecture for all applications. Its built-in variability and flexibility should support the entire range of application architectures. The application architecture is derived by binding the variability of the reference architecture that is documented in the orthogonal variability model. If application-specific requirements make it necessary to adapt the reference architecture, the stakeholders must carefully weigh up the cost and benefit against each other. We deal with the development of the application architecture based on application requirements and the reference architecture in Chapter 16.

2.7.4 Application Realisation Artefacts

Configuration parameters

Application realisation artefacts encompass the component and interface designs of a specific application as well as the configured, executable application itself. The required values for configuration parameters can be pro-

vided, for example, via configuration files. These parameter values are evaluated, for example, by makefiles or the run-time system. The values can be derived from the application variability model.

Application-specific realisation

Many application realisation artefacts are created by reusing domain artefacts and binding the available variability. However, part of the realisation artefacts usually has to be developed in the application realisation subprocess for the specific application. Chapter 17 deals with the development of an application based on reusable components.

2.7.5 Application Test Artefacts

Complete test documentation

Application test artefacts comprise the test documentation for a specific application. This documentation makes application testing traceable and repeatable. Many application test artefacts can be created by binding the variability of domain test artefacts which is captured in the orthogonal variability model. Moreover, detailed test instructions such as the particular input values to be used must be supplemented. For application-specific developments, additional test artefacts must be created. We deal with the development of application test artefacts in Chapter 18.

2.8 Role of the Framework in the Book

Two processes and variability

The book is organised according to the two key differences between software product line engineering and single-system development:

- The need for *two distinct development processes*, namely the domain engineering process and the application engineering process.
- The need to explicitly define and manage *variability*.

Part II chapters

Part II elaborates on the definition of variability, which is the central concept for realising mass customisation in software product line engineering. The part consists of five chapters:

- Principles of Variability (Chapter 4)
- Documenting Variability in Requirements Artefacts (Chapter 5)
- Documenting Variability in Design Artefacts (Chapter 6)
- Documenting Variability in Realisation Artefacts (Chapter 7)
- Documenting Variability in Test Artefacts (Chapter 8)

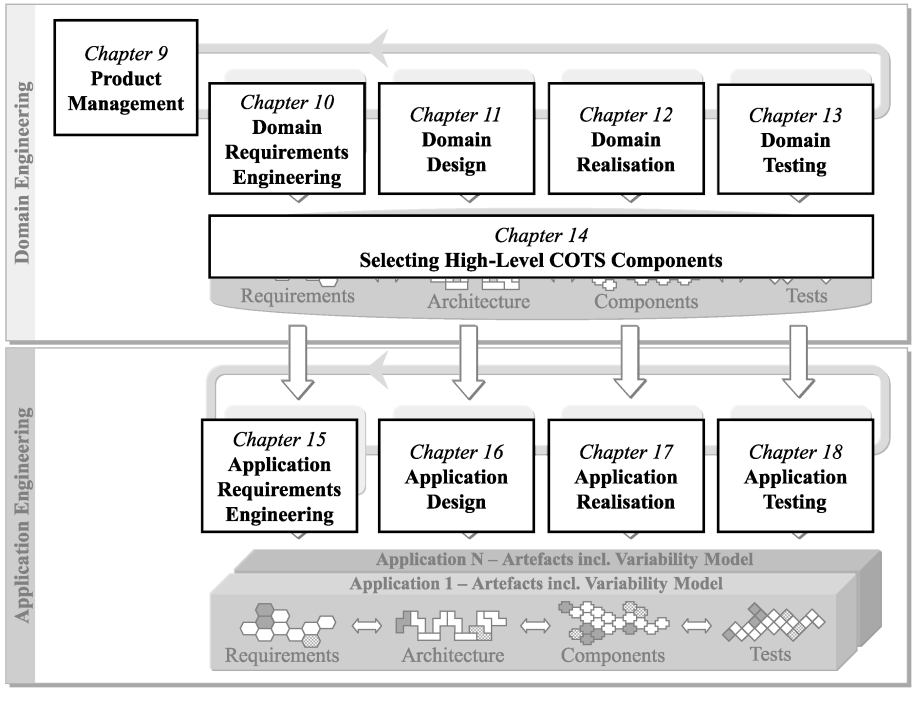

Fig. 2-4: Structure of Parts III and IV

Part III chapters

Part III elaborates on the creation of the platform and thus on the definition of the commonality and the variability of the software product line. The chapters are shown in the upper half of Fig. 2-4. Each of the first five chapters of Part III elaborates on one of the sub-processes of domain engineering (as shown in the upper part of Fig. 2-4). The last chapter of Part III deals with the specific problem of selecting off-the-shelf components in domain engineering. The chapters of Part III are:

- Product Management (Chapter 9)
- Domain Requirements Engineering (Chapter 10)
- Domain Design (Chapter 11)
- Domain Realisation (Chapter 12)
- Domain Testing (Chapter 13)
- Selecting High-Level COTS[11] Components (Chapter 14)

[11] COTS is the acronym for Commercial Off-The-Shelf. A high-level COTS component provides a significant fraction of the functionality of a software product line.

Part IV chapters

Part IV elaborates on the use of the platform to derive specific product line applications. It shows how product line variability is exploited to develop different applications. Each chapter explains one of the four application engineering sub-processes (shown in the lower half of Fig. 2-4):

- Application Requirements Engineering (Chapter 15)
- Application Design (Chapter 16)
- Application Realisation (Chapter 17)
- Application Testing (Chapter 18)

Part V chapters

Part V deals with the institutionalisation of software product line engineering in an organisation. Its focus is on the separation between the domain and application engineering processes. The chapters of Part V are:

- Organisation (Chapter 19)
- Transition Process (Chapter 20).

Part VI chapters

Part VI presents experiences with software product line engineering gained in 15 organisations and briefly characterises essential fields for future research. Whenever possible, we employ the terminology introduced in our framework to make clear the relationships between the topics considered in Part VI and our framework. The chapters of Part VI are:

- Experiences with Software Product Line Engineering (Chapter 21)
- Future Research (Chapter 22)

http://www.springer.com/978-3-540-24372-4

Software Product Line Engineering
Foundations, Principles and Techniques
Pohl, K.; Böckle, G.; van der Linden, F.J.
2005, XXVI, 467 p., Hardcover
ISBN: 978-3-540-24372-4