Script **generated by TTT**

Title: Matthes: Soft-Arch (10.01.2012)

Date: Tue Jan 10 18:15:03 CET 2012

Duration: 62:52 min

Pages: 37

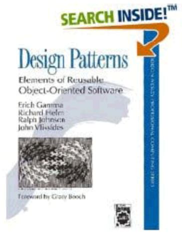
4 – Reuse of Software Architectures

sebis

- Design Patterns
- Architectural Patterns
- Frameworks
- Reference Architectures
- Software Product Line Engineering

Recommended Reading: [Ga95]

sebis



The book describes 23 patterns for managing object creation, composing objects into larger structures, and coordinating control flow between objects.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Design Patterns – Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995

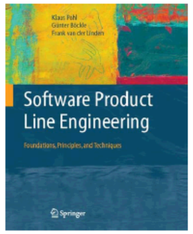
Design Patterns is a modern classic in the literature of object-oriented development, offering timeless and elegant solutions to common problems in software design.

What is a Pattern?

sebis

- Current use comes from the work of the architect **Christopher Alexander**. Alexander studied ways to improve the process of designing buildings and urban areas.
- *"Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."*
- *"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution."*
- Hence, the common definition of a pattern: "A solution to a problem in a context."
- Patterns can be applied to many different areas of human endeavor, including software development.

Recommended Reading: [PBL05]


Klaus Pohl
Günter Böckle
Frank van der Linden
Software Product Line Engineering
Foundations, Principles, and Techniques
Springer

Software product line engineering has proven to be the paradigm for developing a diversity of software products and software-intensive systems in shorter time, at lower cost, and with higher quality. With more than 100 examples and about 150 illustrations, the authors describe in detail the essential foundations, principles and techniques of software product line engineering.

The authors are professionals and researchers who significantly influenced the software product line engineering paradigm and successfully applied software product line engineering principles in industry.

Pohl, K.; Böckle, G.; Linden, F.: *Software Product Line Engineering.* Springer, 2005.

This textbook addresses students, professionals, lecturers and researchers interested in software product line engineering.

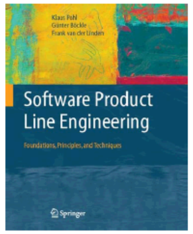
Recommended Reading: [PBL05]

Software product line engineering has proven to be the paradigm for developing a diversity of software products and software-intensive systems in shorter time, at lower cost, and with higher quality. With more than 100 examples and about 150 illustrations, the authors describe in detail the essential foundations, principles and techniques of software product line engineering.

The authors are professionals and researchers who significantly influenced the software product line engineering paradigm and successfully applied software product line engineering principles in industry.

Pohl, K.; Böckle, G.; Linden, F.: *Software Product Line Engineering.* Springer, 2005.

This textbook addresses students, professionals, lecturers and researchers interested in software product line engineering.


Klaus Pohl
Günter Böckle
Frank van der Linden
Software Product Line Engineering
Foundations, Principles, and Techniques
Springer

Terminology

- "Software product family" and "software product line" are used almost synonymously.

Traditionally: two kinds of products:

- Handcrafted for individual customers
- Mass production using **production lines**

Software domain: individual vs. standard software

A rising demand for individualized products lead to **mass customization**:

Mass customization is the large-scale production of goods tailored to individual customer's needs. [Dav87]

Platforms

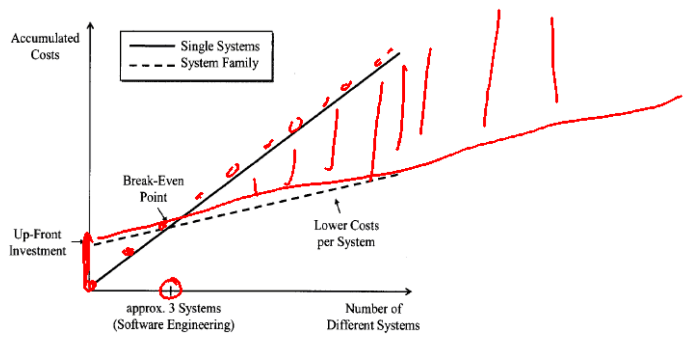
In order to achieve mass customization, companies started to introduce **common platforms** for different types of products.

Example from car manufacturing:

- A platform provides a structure for major components determining the body size, and the size and type of the engine and transmission.
- The parts comprising the platform were usually the most expensive subsystems in terms of design and manufacturing preparation costs
- The use of the platform for different car types typically led to a reduction in the production cost for a particular car type

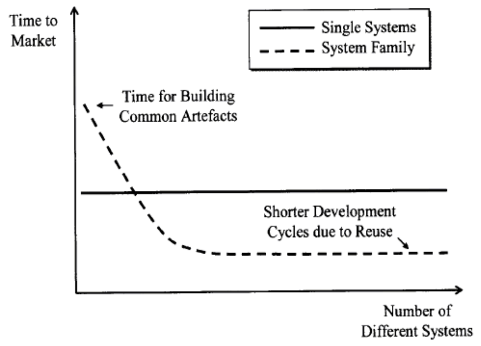
Motivation for Product Line Engineering (1)

- Reduction of development costs



Motivation for Product Line Engineering (2)

- Reduction of time to market
- Enhancement of quality
- Coping with evolution
- Coping with complexity
- Improving cost estimations



Definition of Software Product Line Engineering

Software product line engineering is a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customization.

- Developing applications using platforms means to plan proactively for reuse.
- ~~Building applications~~ for mass customization means employing the concept of **managed variability** to model the commonalities and the differences in the applications in a systematic way.

Model-based

Software Platform

In the software industry the term **platform** is often used to denote the underlying computer system on which applications are developed and deployed, e.g. operation systems, processor families, software runtime environments.

In the context of software product line engineering the term **platform** has to reflect the creation of entire products from reusable parts.

Definition:

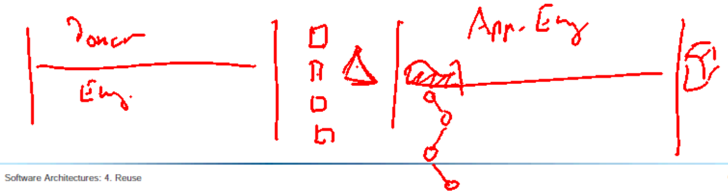
A **software platform** is a set of software subsystems and interfaces that form a common structure from which a set of derivative products can be efficiently developed and produced. [ML97]

Two Development Processes

The software product line engineering paradigm separates two processes:

- **Domain engineering**: This process is responsible for establishing the reusable platform and thus for defining the commonality and the variability of the product line.
- **Application engineering**: This process is responsible for deriving product line applications from the platform by reusing domain artifacts and exploiting the product line variability.

- The two process must interact in a manner that is beneficial to both.
- A large part of application engineering consists of reusing the platform and binding the variability as required for the different applications.

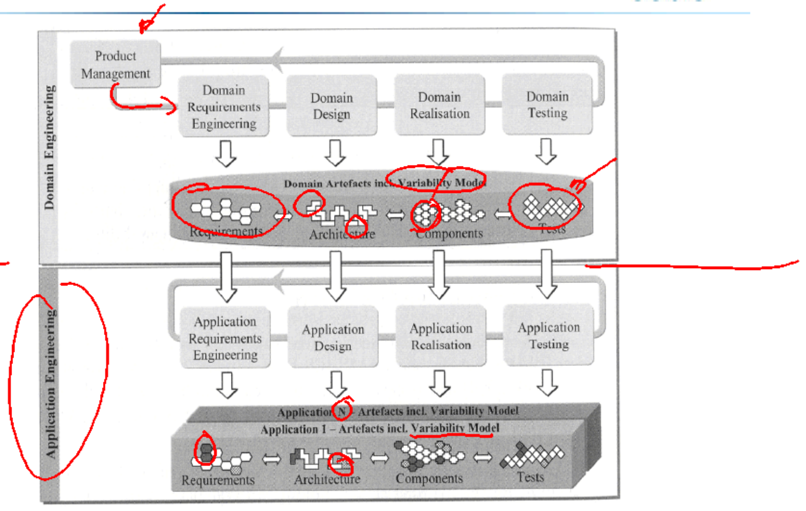


Domain Engineering

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 artifacts that accomplish the desired variability.

The Software Product Line Engineering Framework



Application Engineering

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 applications.
- Exploit the commonality and the variability of the software product line during the development of a product line application.
- Document the application artifacts, i.e. application requirements, architecture, components, and tests, and relate them to the domain artifacts.
- 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.

Managed Variability

Variability is defined during domain engineering, it is exploited during application engineering.

Defining and exploiting variability is supported by the concept of **managed variability**:

- Supporting variability concerned with defining variability.
- Managing variable artifacts.
- Supporting activities concerned with resolving variability.
- Collecting, storing and managing trace information necessary to fulfill these tasks.

The moment of variability resolution in realization is called the **binding time**.

Variability Subject and Variability Object

What does vary?

- A variability subject is a variable item of the real world or a variable property of such an item.

Why does it vary?

- Different stakeholder needs, different country laws, technical reasons, …

How does it vary?

- A variability object is a particular instance of a variability subject.

Example:

- "Color" is a variability subject of the real world. Examples of variability objects are red, green, blue, …
- "Payment method" is a variability subject and payment by credit card, payment by bill, payment by cash are examples of variability objects.

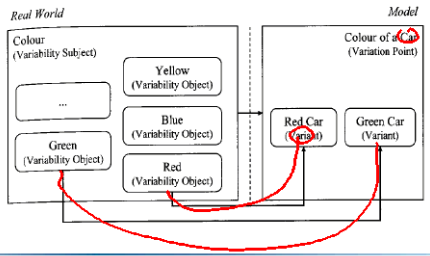
Variability in Software Product Line Engineering

In software product line engineering, variability subjects and the corresponding variability objects are embedded into the context of a software product line.

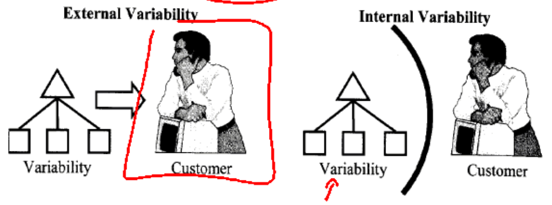
Definition:

A **variation point** is a representation of a variability subject within domain artifacts enriched by contextual information.

A **variant** is a representation of a variability object within domain artifacts.


Real World
Model
Colour (Variability Subject)
Yellow (Variability Object)
...
Blue (Variability Object)
Green (Variability Object)
Red (Variability Object)
Colour of a Car (Variation Point)
Red Car (Variant)
Green Car (Variant)

Internal and External Variability



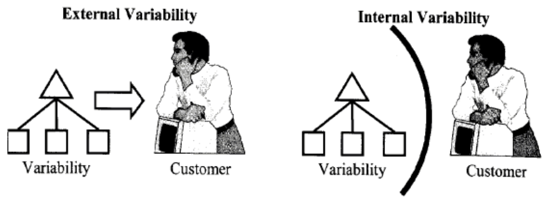
External variability is the variability of domain artifacts that is visible to customers.

- Example: The customers of a home automation system can choose between different door lock identification mechanisms: keypad, magnetic card, and fingerprint scanners.
- Causes: Stakeholder needs, laws and standards

Internal variability is the variability of domain artifacts that is hidden from customers.

- Example: The communication protocol of a home automation system network offers two different modes.
- Causes: Refinement of external variability; technical reasons

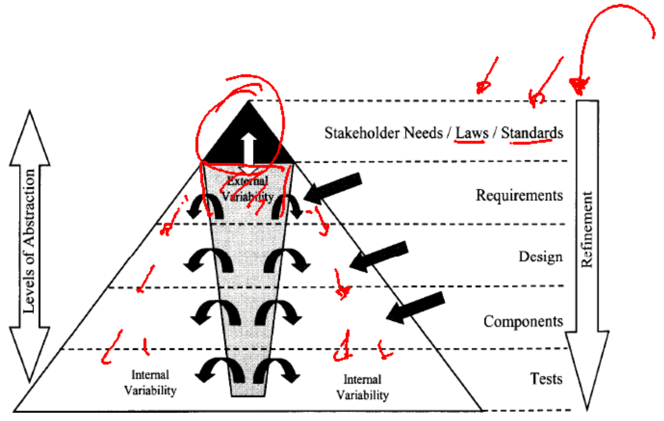
Internal and External Variability



External variability is the variability of domain artifacts that is visible to customers.
- Example: The customers of a home automation system can choose between different door lock identification mechanisms: keypad, magnetic card, and fingerprint scanners.
- Causes: Stakeholder needs, laws and standards

Internal variability is the variability of domain artifacts that is hidden from customers.
- Example: The communication protocol of a home automation system network offers two different modes.
- Causes: Refinement of external variability; technical reasons

The Variability Pyramid



Explicit Documentation of Variability

An adequate documentation of variability information should at least include all the information needed to answer the following questions:
- What varies?
  - Variable properties of the development artifacts have to be documented by variation points.
- Why does it vary?
  - Causes for internal/external variability
- How does it vary?
  - Available variants and their linking to the domain model elements
- For whom is it documented?
  - Audience of a variation point?

Advantages of explicit documentation

- Improves decision making by forcing engineers to document the rationales for introducing a certain variation point
- Improves communication about the variability of a software product line by providing a high-level abstraction of variable artifacts
- Allows for improved traceability of variability between its source and the corresponding variable artifacts.

Variability Definition

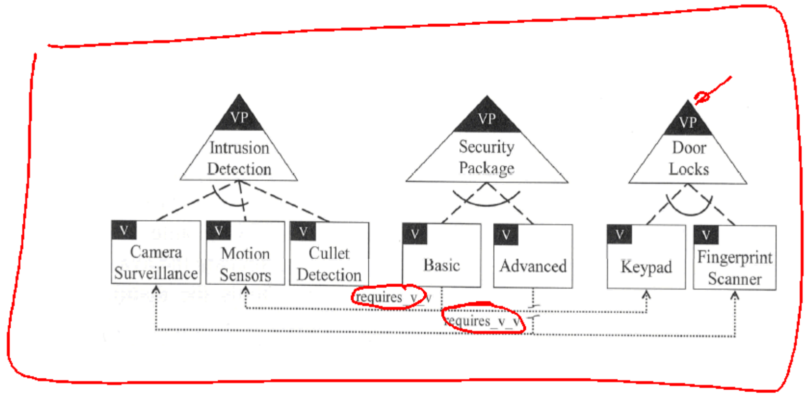
Variability can be defined either as an integral part of development artifacts or in a separate variability model.

Shortcomings of modeling variability within the traditional software development models:

- If variability is spread, it is almost impossible to keep the information consistent
- It is hard to determine, which variability information in requirements has influenced which variability information in design, realization or test artifacts.
- The software development models are already complex and get overloaded by adding the variability information
- The concepts used to define variability differ between the different kinds of software development models

→ The variability defined in different models does not integrate well into an overall picture of the software variability.

Example (2)

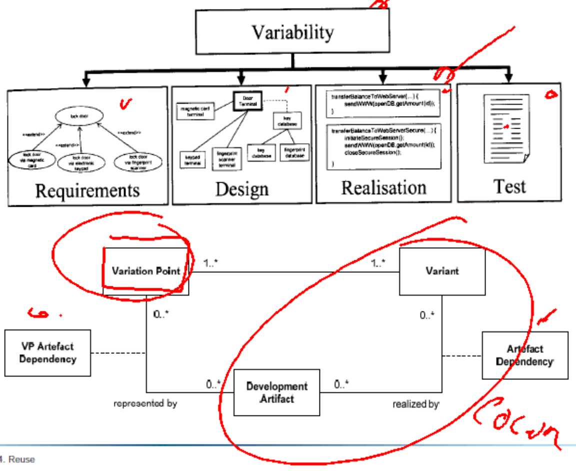


- Design Patterns
- Architectural Patterns
- Frameworks
- Reference Architectures
- Software Product Line Engineering
  - Principles of Product Line Engineering
  - A Framework for Software Product Line Engineering
  - Principles of Variability
  - Documenting Variability

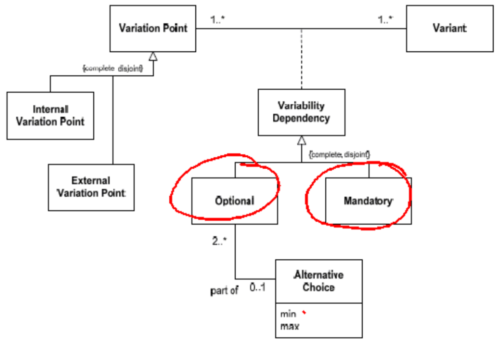
6

Traceability between Variability Model and Other Development Artifacts

Variability defined in the variability model has to be related to software artifacts specified in other models, textual documents, and code

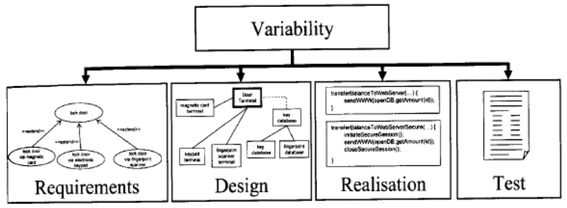


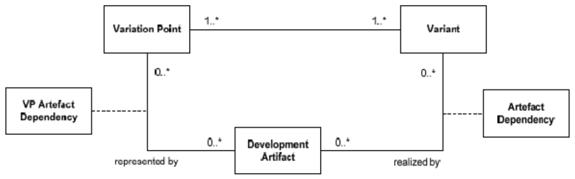
Variability Meta Model



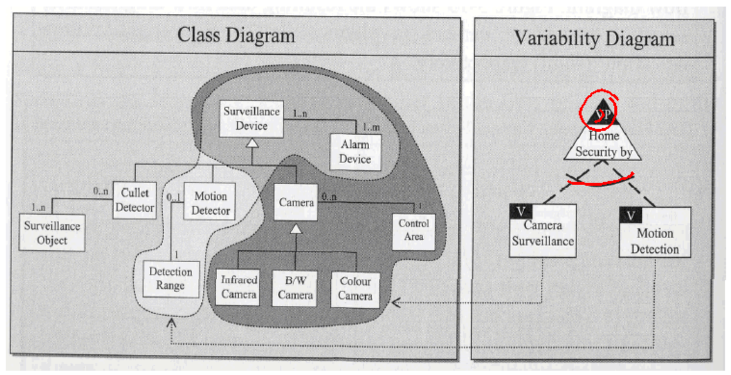
Traceability between Variability Model and Other Development Artifacts

Variability defined in the variability model has to be related to software artifacts specified in other models, textual documents, and code

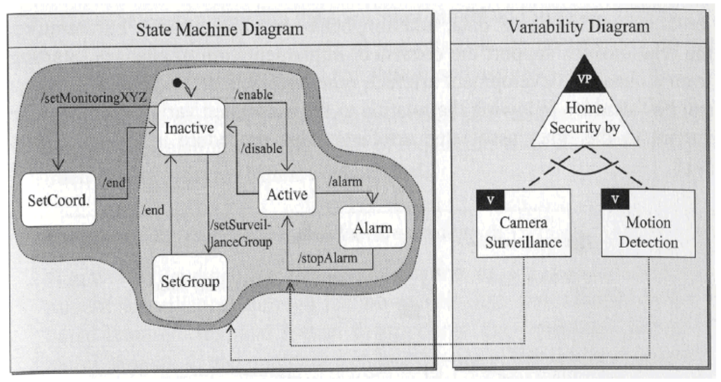




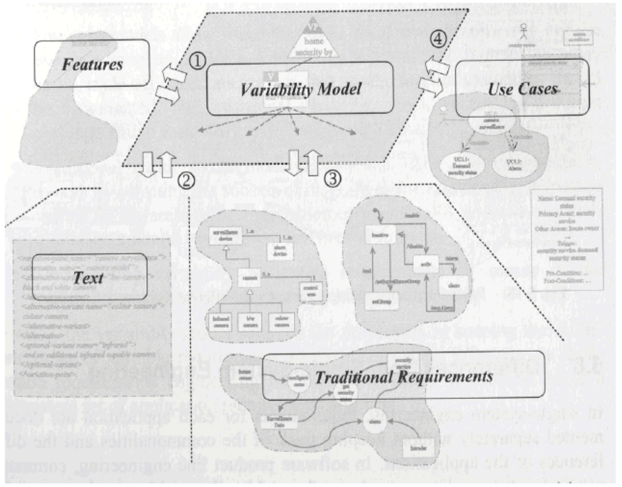
Documenting Variability in a Class Diagram



Documenting Variability in a State Diagram



Relationships between Requirements Artifacts and the Variability Model



Conclusion

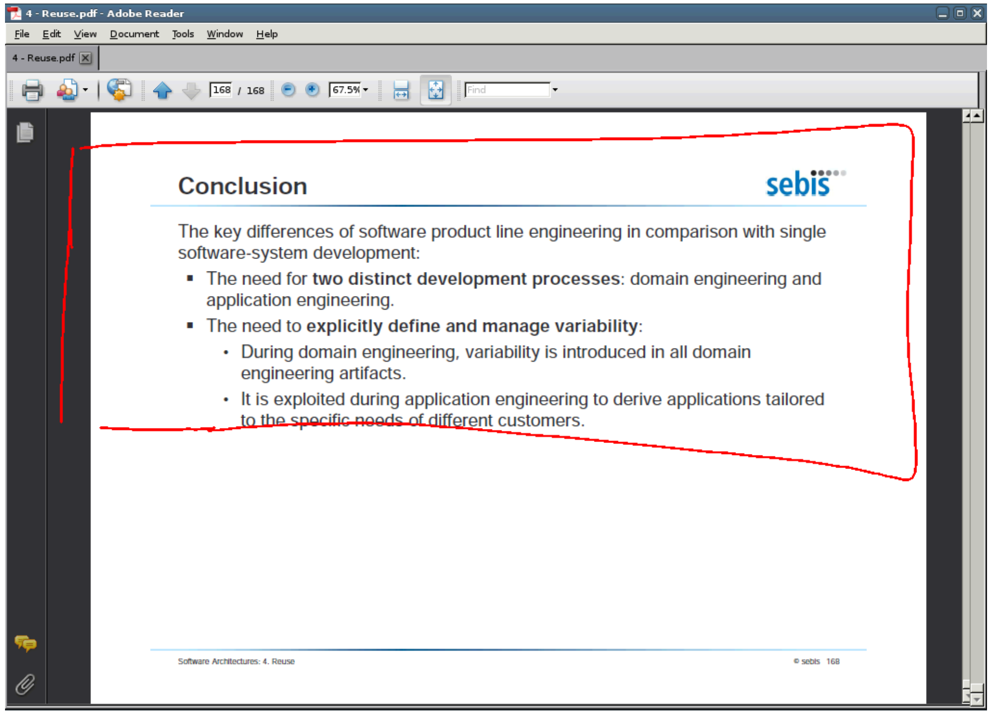
The key differences of software product line engineering in comparison with single software-system development:

- The need for **two distinct development processes**: domain engineering and application engineering.
- The need to **explicitly define and manage variability**:
  - During domain engineering, variability is introduced in all domain engineering artifacts.
  - It is exploited during application engineering to derive applications tailored to the specific needs of different customers.

Conclusion

The key differences of software product line engineering in comparison with single software-system development:

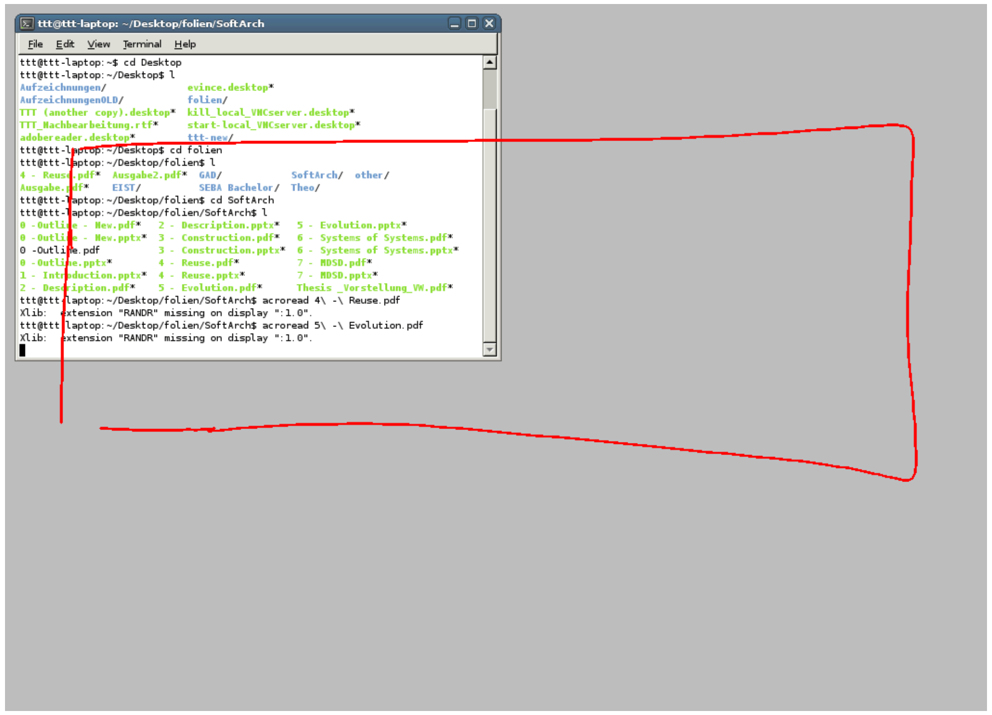
- The need for **two distinct development processes**: domain engineering and application engineering.
- The need to **explicitly define and manage variability**:
  - During domain engineering, variability is introduced in all domain engineering artifacts.
  - It is exploited during application engineering to derive applications tailored to the specific needs of different customers.


4 - Reuse.pdf - Adobe Reader
File Edit View Document Tools Window Help
4 - Reuse.pdf
168 / 168
67.5%
Find

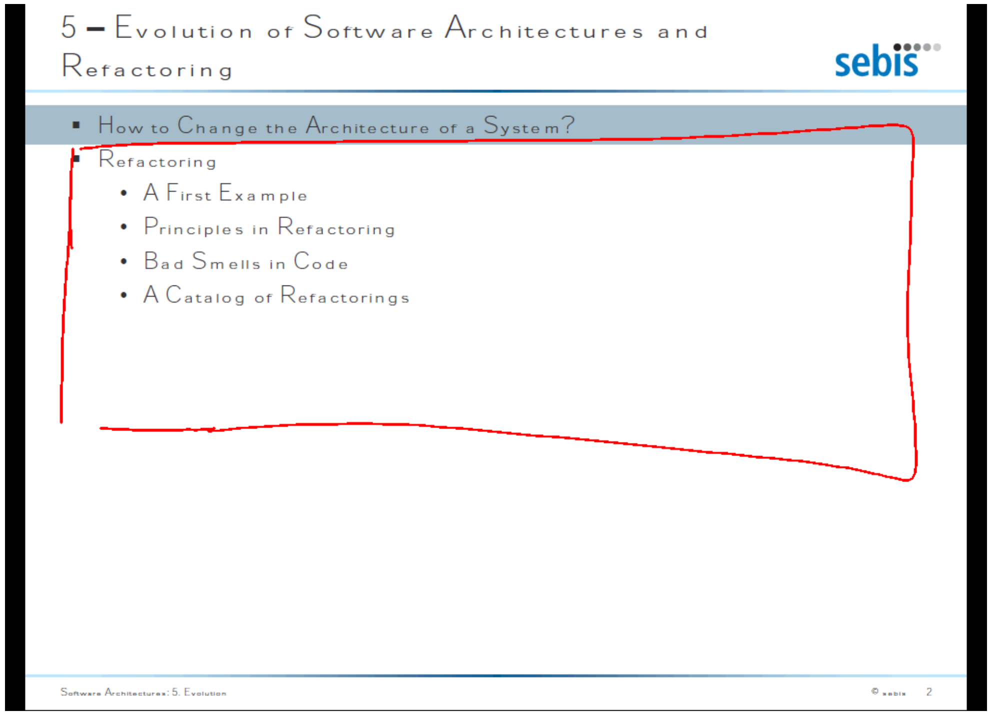
Conclusion

The key differences of software product line engineering in comparison with single software-system development:

- The need for **two distinct development processes**: domain engineering and application engineering.
- The need to **explicitly define and manage variability**:
  - During domain engineering, variability is introduced in all domain engineering artifacts.
  - It is exploited during application engineering to derive applications tailored to the specific needs of different customers.


ttt@ttt-laptop: ~/Desktop/folien/SoftArch
File Edit View Terminal Help

```
ttt@ttt-laptop:~$ cd Desktop
ttt@ttt-laptop:~/Desktop$ l
Aufzeichnungen/              evince.desktop*
AufzeichnungenOLD/           folien/
TTT (another copy).desktop*  kill_local_VNCserver.desktop*
TTT_Nachbearbeitung.rtf*     start-local_VNCserver.desktop*
adobereader.desktop*         ttt-new/
ttt@ttt-laptop:~/Desktop$ cd folien
ttt@ttt-laptop:~/Desktop/folien$ l
4 - Reuse.pdf*  Ausgabe2.pdf*  GAD/            SoftArch/  other/
Ausgabe.pdf*    EIST/          SEBA Bachelor/  Theo/
ttt@ttt-laptop:~/Desktop/folien$ cd SoftArch
ttt@ttt-laptop:~/Desktop/folien/SoftArch$ l
0 -Outline - New.pdf*   2 - Description.pptx*   5 - Evolution.pptx*
0 -Outline - New.pptx*  3 - Construction.pdf*   6 - Systems of Systems.pdf*
0 -Outline.pdf          3 - Construction.pptx*  6 - Systems of Systems.pptx*
0 -Outline.pptx*        4 - Reuse.pdf*          7 - MDSD.pdf*
1 - Introduction.pptx*  4 - Reuse.pptx*         7 - MDSD.pptx*
2 - Description.pdf*    5 - Evolution.pdf*      Thesis _Vorstellung_VW.pdf*
ttt@ttt-laptop:~/Desktop/folien/SoftArch$ acroread 4\ -\ Reuse.pdf
Xlib:  extension "RANDR" missing on display ":1.0".
ttt@ttt-laptop:~/Desktop/folien/SoftArch$ acroread 5\ -\ Evolution.pdf
Xlib:  extension "RANDR" missing on display ":1.0".
█
```



5 – Evolution of Software Architectures and Refactoring

sebis

- How to Change the Architecture of a System?
- Refactoring
  - A First Example
  - Principles in Refactoring
  - Bad Smells in Code
  - A Catalog of Refactorings