

A faint UML class diagram is visible in the background. It shows several classes: 'Person' with attributes 'id: int', 'name: String', and 'dept: String'; 'Address' with attributes 'id: int', 'name: String', and 'zip: String'; and 'Department' with attributes 'id: int' and 'name: String'. There are associations between 'Person' and 'Address' (labeled 'address' with multiplicity '0..1'), and between 'Person' and 'Department' (labeled 'dept' with multiplicity '0..1').

An overview of requirements engineering for software product lines

Requirements Engineering II

December 1st 2008

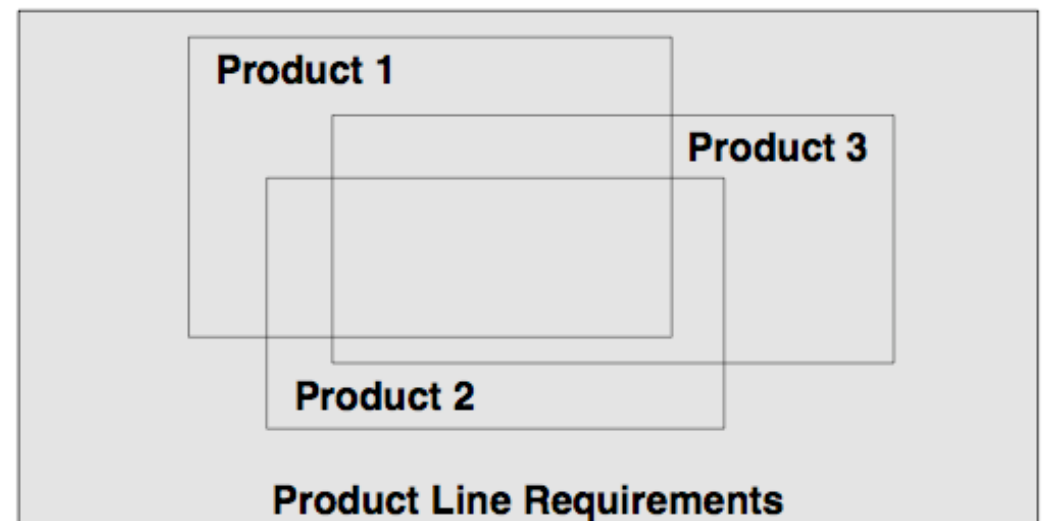
Michael Sommerhalder, Krzysztof Dabkowski, Stefan Schurgast

Agenda

- What are software product lines?
- General overview
- Processes – domain and application analysis
- Advantages and challenges

Definition

”A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy needs of a particular market or mission and that are developed from a common set of core assets in a prescribed way”



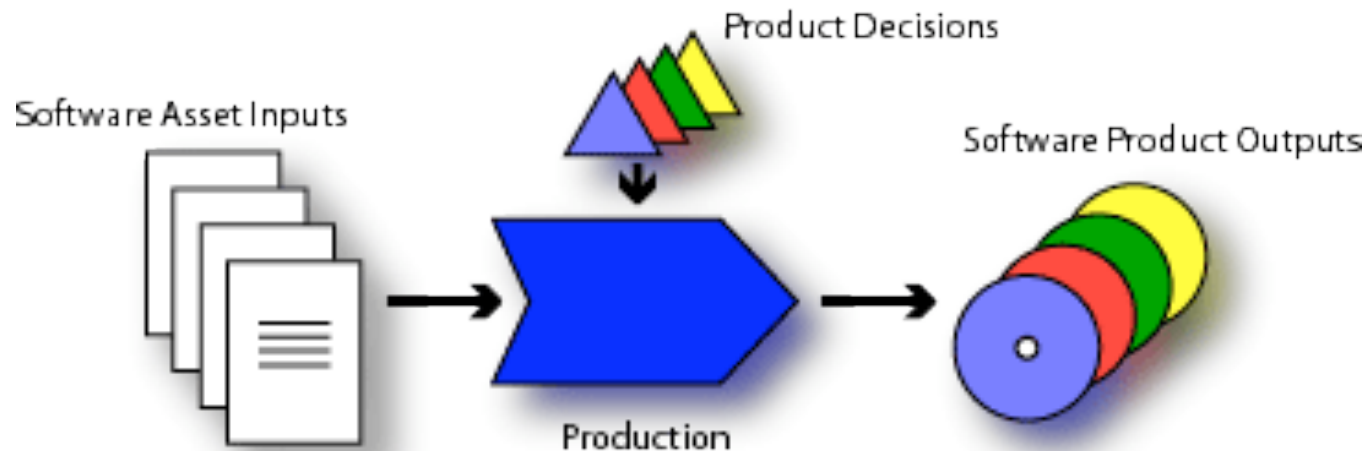
[Chastek 2001]

Rationale

- Predictive software reuse:
 - ◆ Only create software artefacts if reuse is predicted
 - ◆ Not putting them into a library and hoping to reuse them at some point in the future (opportunistic approach)
- Mass customisation:
 - ◆ Being able to efficiently create many variations of a product
 - Needs focus on communalities
 - And effective management of variations in product line

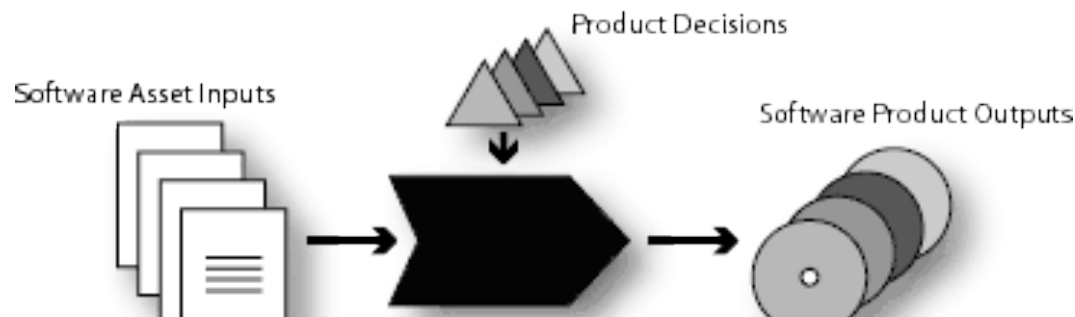
General overview

- Four basic parts:
 - ◆ Software Asset Inputs
 - ◆ Decision model and product decisions
 - ◆ Production mechanism and process
 - ◆ Software product outputs



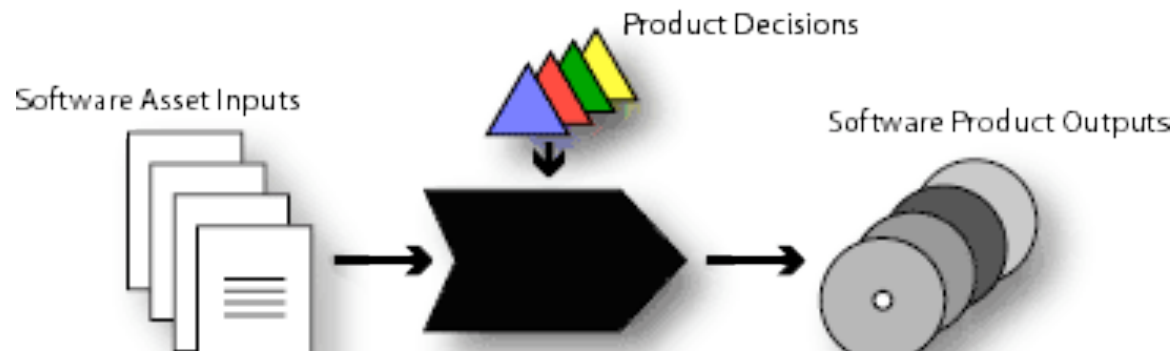
Software Asset Inputs

- Representation
 - ◆ Requirements, source code components, test cases, architecture, and documentation, e.g.
- Variation Points
 - ◆ Can be configured in different ways to achieve different behaviour
 - ◆ When to configure them is influenced by desired binding time/production automation



Decision model

- Decision model
 - ♦ Describes optional and variable features for the products in the product line.
- Product Decisions
 - ♦ Choices for each of the optional and variable features in the decision model. Defines each product uniquely

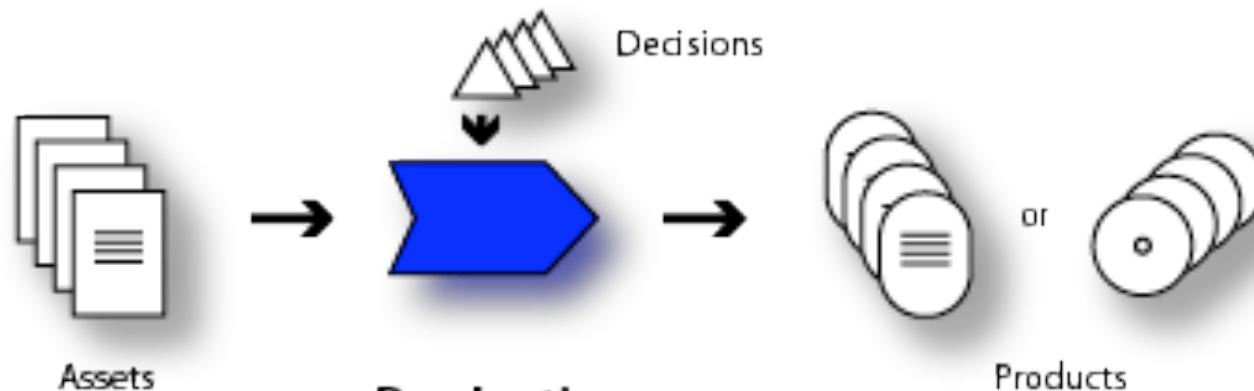


Decision model – 2

- Representation
 - ♦ Different formats range from textual descriptions to formal languages
- Guidance
 - ♦ Some approaches provide guidance when making decisions, e.g. constraint checking
- Role
 - ♦ Decision-maker might be an engineer or non-engineer. Shortest time of decision is needed.
- Replayable
 - ♦ In periodic production, previous decisions can be "replayed" instead of manually created

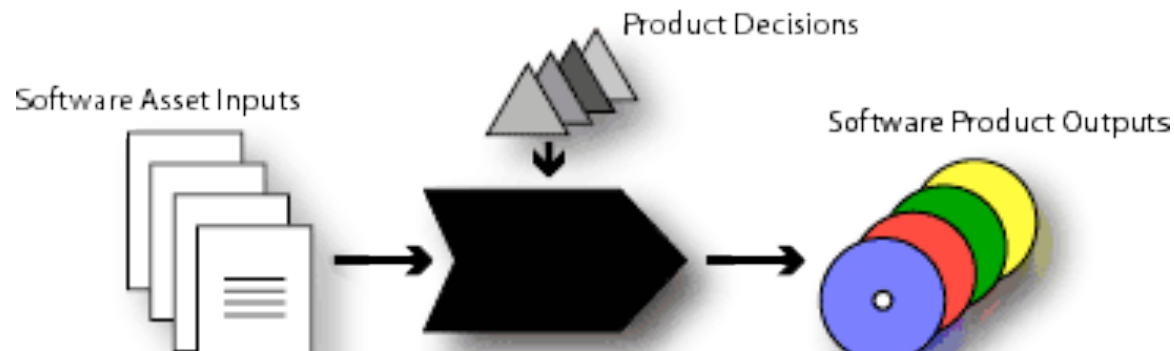
Production

- Composes and configures products from the software asset inputs using product decisions
- Approaches different in:
 - ♦ Automation: Fully automated, fully manual, in between
 - ♦ Periodicity: Frequency of periodic production
 - ♦ Roles: Separate human role for application engineering, distinct from domain engineering possible but not mandatory



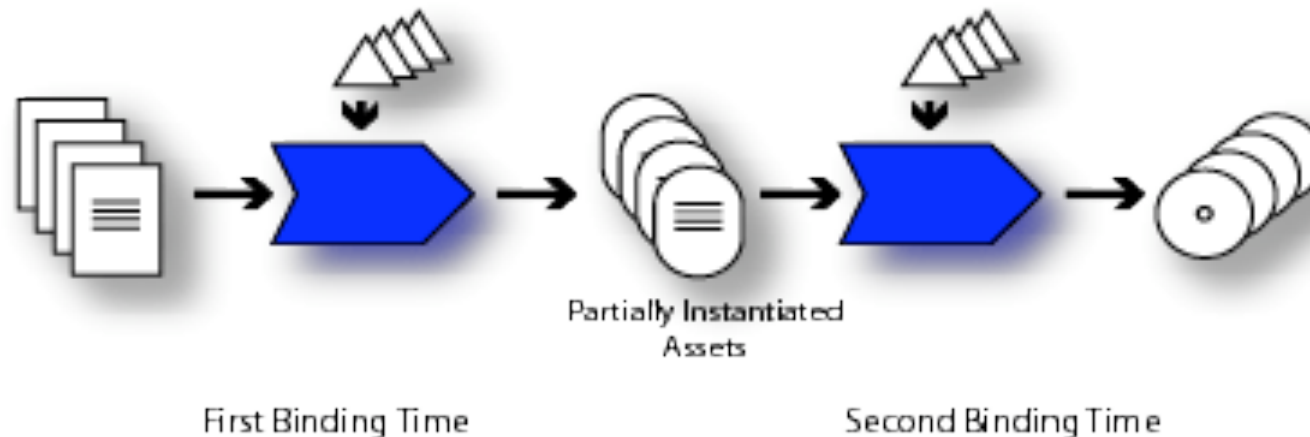
Software Product Outputs

- The collection of all products that can be produced for the product line.
- Representation
 - ♦ Many different formats, binary/source, e.g.
- Partial Instantiation
 - ♦ Variation points in software asset inputs for next stage



Binding times

- Presence of variation in assets is primary distinction between software product lines and conventional software engineering
- Assets contain variation points (unbound options how the software will behave) in early stages of software product lines lifecycle
- Binding time: Time at which the variation is bound



Domain engineering process

- Objectives:
 1. Establishment of commonalities and variability for a software product line
 2. Defining the scope of the software product line – a set of applications the product is planned for
- Sub processes:
 1. Product management
 - Employs scoping techniques (what is within a scope)

Domain engineering process – 2

2. Domain Requirements Engineering

- Elicitation and documentation of common and variable requirements

3. Domain Design

- Design reference architecture – a high-level structure of all product line applications

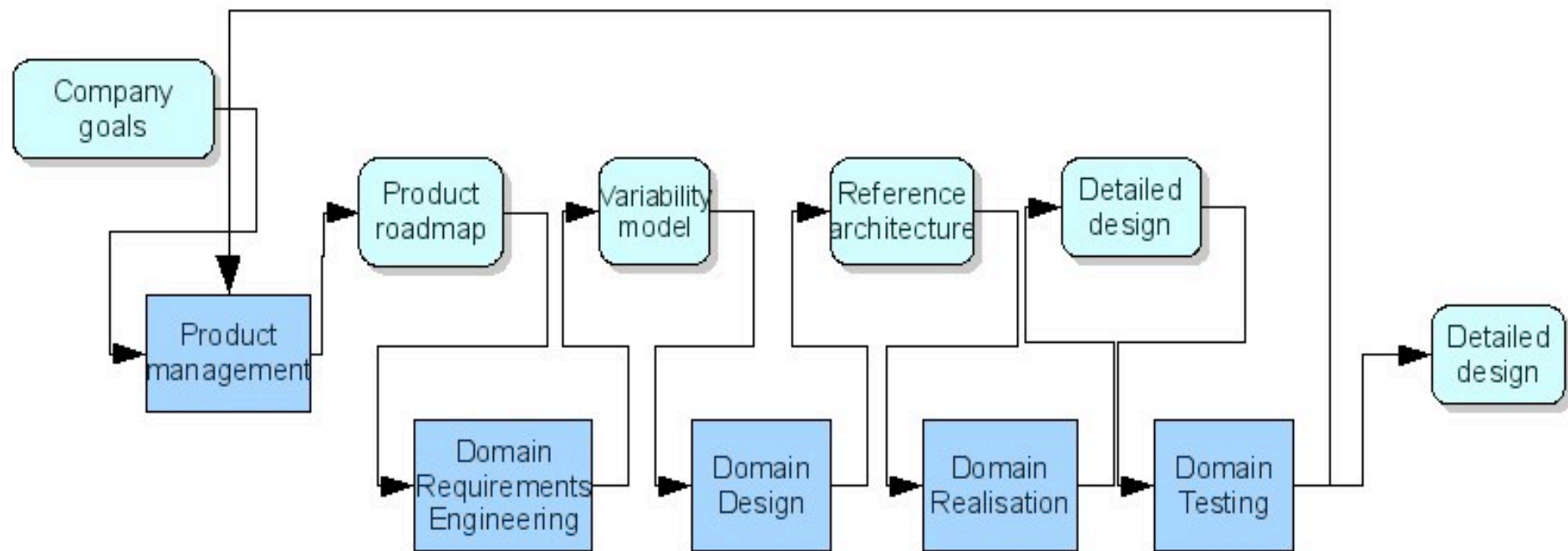
4. Domain Realisation

- Design and implementation of reusable software components

5. Domain Testing

- Validation and verification of components

Domain engineering process – 3



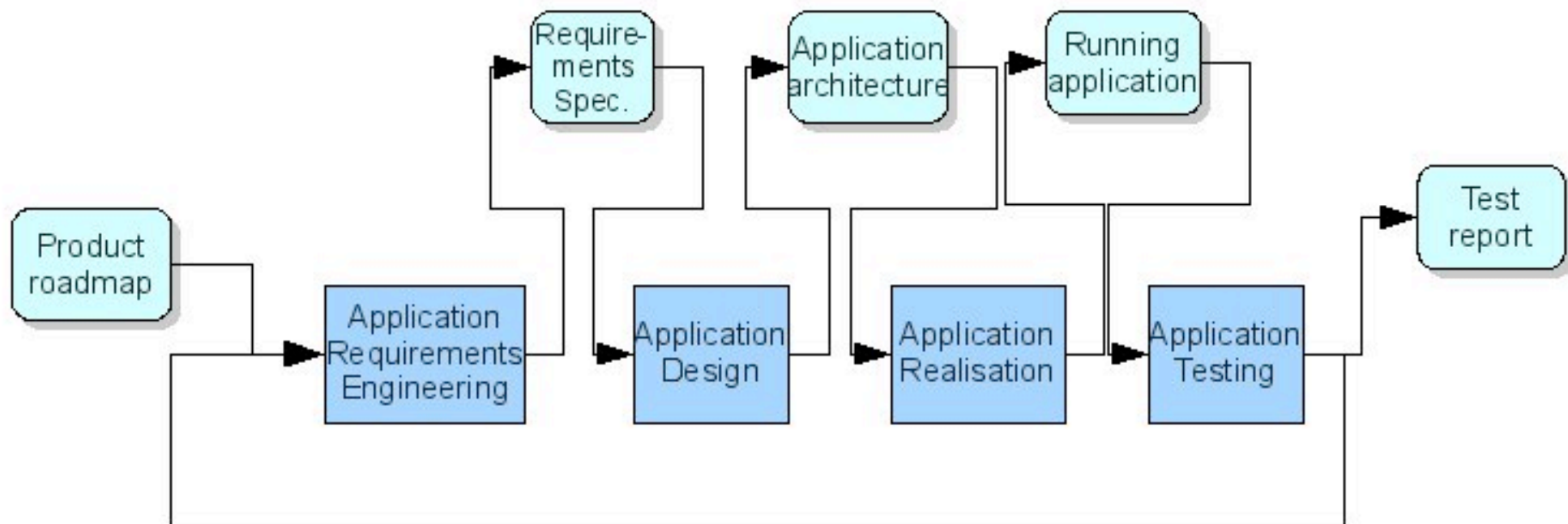
Application engineering process

- Objectives:
 1. High as possible reuse of the domain assets during the development
 2. Exploiting commonalities and variability during the product development
 3. Document the application artefacts

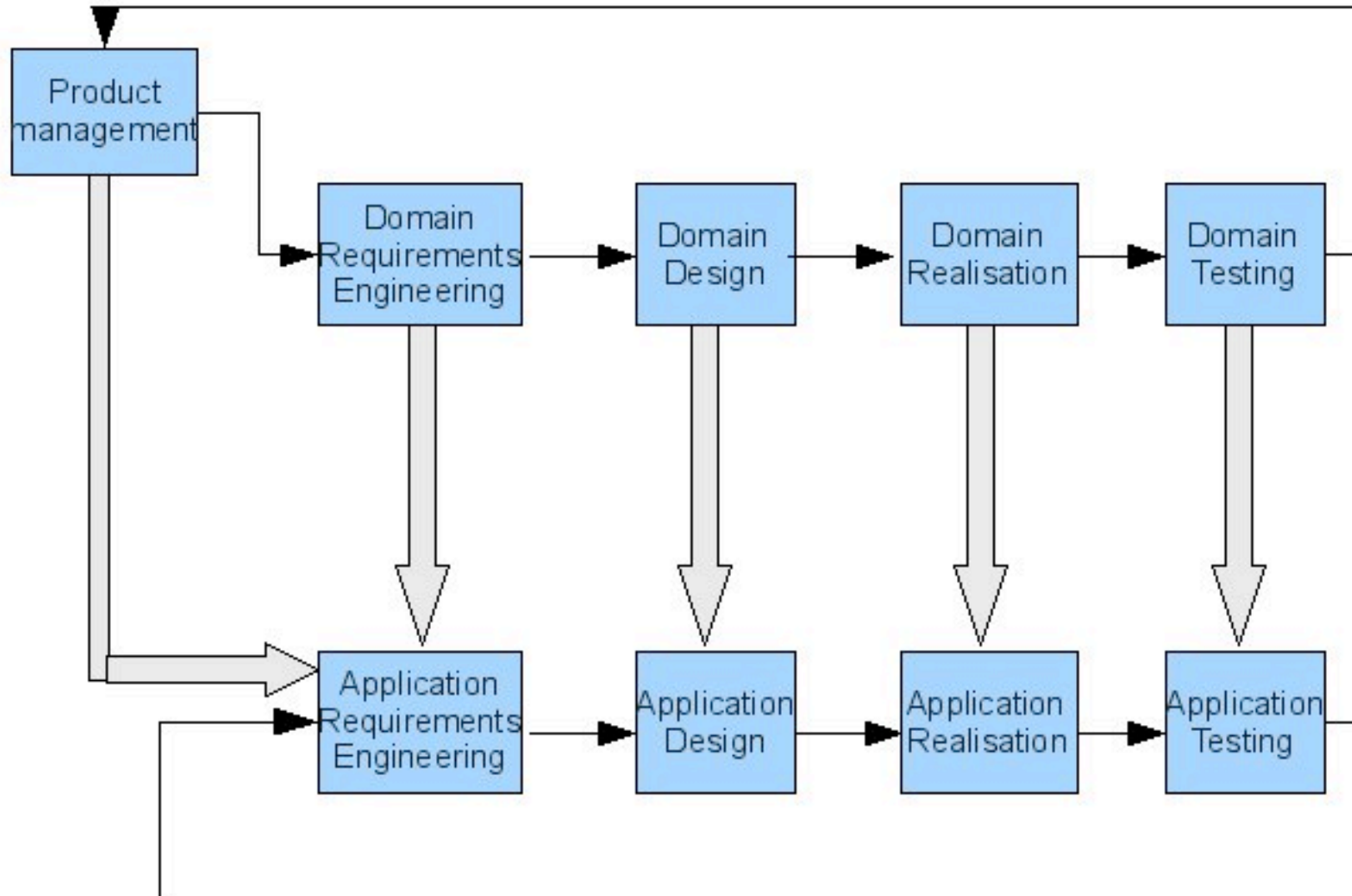
Application engineering process – 2

- Sub processes:
 1. Application Requirements Engineering
 - Development of the application requirements specification
 2. Application Design
 - Instantiating of the application architecture from the reference architecture, incorporate application specific adaptations
 3. Application Realisation
 - Selection and configuration of reusable components
 4. Application Testing
 - Validation and verification against the specification

Application engineering process – 3



Domain and Application engineering



Advantages

- Strategic application yields order of magnitude improvements due to reusability of ideas in
 - ◆ Time-to-market
 - ◆ Quality
 - ◆ Portfolio scalability
 - ◆ Software engineering cost

Challenges

- Reusability of components and configuration management
- Complexity of domain models rises

Literature

[Pohl 2005]: Pohl, K., G. Böckle, and F.van der Linden (2005). Software Product Line Engineering – Foundations, Principles, and Techniques. Heidelberg: Springer.

[Chastek 2001]: Chastek, G., P. Donohoe, K.C. Kang, and S. Thiel (2001). Product Line Analysis: A Practical troduction. CMU/SEI Technical Report CMU/SEI-2001-TR-001.

[<http://www.softwareproductlines.com/>]:
<http://www.softwareproductlines.com/introduction/>

Thanks for your attention!

