Software Product Lines

Software Product Line Engineering and Architectures

Bodo Igler and Burkhardt Renz

Hochschule Rhein-Main and Technische Hochschule Mittelhessen

Wintersemester 2017/18

Questions

- How can you produce many different but related software products? (mass production)
- How can you do this,
  - if you have to satisfy special customer requirements? (customization)
  - if the products have to be cheap *and* good? (cost efficieny, quality)
  - if you have to react quickly to changing requirements? (time to market)

Answer: Adopt a Software Product Line Approach.

Table of Contents

- Introduction
  - Examples
  - Terminology
  - Approach
- Commonality and Variability
  - Motivation
  - Commonality and Variability Analysis
  - Features
- Putting Things Together
  - Architecture
  - The Big Picture
  - Conclusion

Overview

- Introduction
  - Examples
  - Terminology
  - Approach
- Commonality and Variability
  - Motivation
  - Commonality and Variability Analysis
  - Features
- Putting Things Together
  - Architecture
  - The Big Picture
  - Conclusion

Example

Car Manufacturing

Example

Integrated Development Environment

Product Lines

Mass Customization

- mass production
- customization

Platform

- base of technologies
- other technologies use this base

Product Line

Family of products which share common features (*commonalities*).

Software Product Lines

Software Platform

- set of software subsystems and interfaces
- common structure
- facilitates efficient development and production of derivative products
- comprises several artifacts
  - code
  - architecture
  - requirements
  - manuals
  - test cases
  - ...

Software Product Line Engineering

Software Product Line

> *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 segment or mission and that are developed from a common set of core assets in a prescribed way.*
>
> –Paul Clements, Linda Northrop

Software Product Line Engineering

- develop family of software applications
- apply mass customization
- use software platform

Overall Process

Software Product Line Engineering =

Domain Engineering
“produce the platform”

= requirements

+ design

+ implementation

+ test

+

Application Engineering
“produce a single product”

= requirements

+ design

+ implementation

+ test

Domain Engineering

Results

- definition of commonality

  "What is common to all products?"
- definition of variability

  "What is different? What is allowed to vary?"

  "How does it vary? How is it allowed to vary?"

⇒ platform = reusable artifacts (*domain artifacts*, "skeleton")

During Each Step

- detail variability from previous step
- add – if necessary – internal variability

Application Engineering

Results

- the product (*application artifacts*, “skeleton + flesh”)
- feedback to domain engineering

During Each Step

bind variability of each domain artifact
⇒ obtain application artifacts

- fill in templates
- implement interfaces
- provide configuration files
- ...

Overview

- Introduction
  - Examples
  - Terminology
  - Approach

- Commonality and Variability
  - Motivation
  - Commonality and Variability Analysis
  - Features

- Putting Things Together
  - Architecture
  - The Big Picture
  - Conclusion

Motivation

Questions:

How do I find the appropriate commonalities?

How do I find the appropriate variability?

Answer: Commonality and Variability Analysis.

Question:

How do I document commonalities and variability?

Answer: Feature Model.

General Idea

Input

variants of one product = product family (Parnas 1976)

Process

1. Commonality Analysis:

   find commonalities

   categorize commonalities

2. Variability Analysis:

   find special properties

   categorize special properties

Output

appropriate abstraction

Examples

Example

IDE Requirements

Example

Database Frontend

Terminology

(Positive) Variability

common degree of freedom

Negative Variability

a degree of freedom is violated under certain circumstances

External Variability

required by and/or visible to customer

Internal Variability

neither required by nor visible to customer

Terminology

Variation Point

something that varies, a degree of freedom
e.g. color, payment method

Variant

potential property of something that varies
e.g. “red”, “green” or “credit card”, “cash”

Binding

fix a variation point by specifying/instantiating a (legal) variant

Binding Time

e.g. design, coding, compilation, installation, run-time

Features

Question:

Why doesn't UML do the job?

Answer:

Standard UML shows *one* model.

We have to show all relevant variations.

Question:

What can do the job?

Answer:

Feature Model.

Features

Feature

“end-user visible characteristic of a system”

Composed Feature

composition of sub-features

Atomic Feature

cannot be divided into sub-features

Feature Model Requirements

A Feature Model is

- to represent all features
- to represent the relationships between features
- to distinguish between commonality and variability
- to be independent of implementation technology
- to be suitable during requirements engineering, design, code and test

Feature Model

A Feature Model comprises

- set of features
- set of feature constraints, usually:
  - type of composition:
    mandatory, optional, alternative, logical or
  - any logical formula with features as atoms:
    $\text{feature}_1 \vee \text{feature}_2 \rightarrow \text{feature}_3, \ldots$

Feature Diagram

Remark

- there are different feature diagram types
- there is no standard available yet
- feature diagrams can be connected to standard UML diagrams

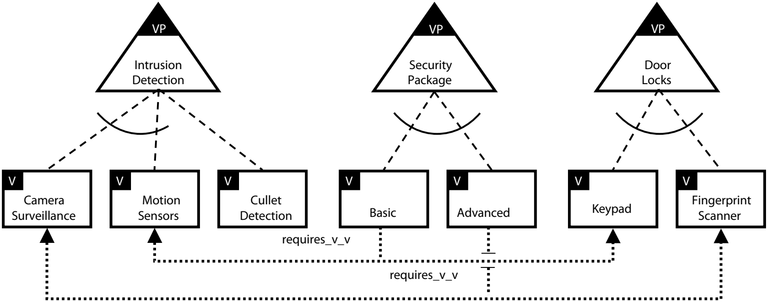
Typical Approach

- diagram = tree
- feature = node
- relationship = edge

Example

Example

orthogonal variability model (Pohl, Böckle, van der Linden et al)



Overview

- Introduction
  - Examples
  - Terminology
  - Approach

- Commonality and Variability
  - Motivation
  - Commonality and Variability Analysis
  - Features

- Putting Things Together
  - Architecture
  - The Big Picture
  - Conclusion

Adapt Functionality I

Use the Template Method Pattern

- platform = application framework + base classes/interfaces
- variation points = (abstract) methods of base classes/interfaces
- bind variation points = provide specific method implementations
- binding time = compile-time
- example: MFC document-view architecture

Adapt Functionality II

Use a Plug-In Architecture

- platform = framework + basic plug-ins
- variation points = extension points of basic plug-ins
- bind variation points = provide specific plug-ins
- binding time = run-time
- example: Eclipse 3.x based on OSGi

Adapt Domain Model

Use a Domain Specific Language (DSL)

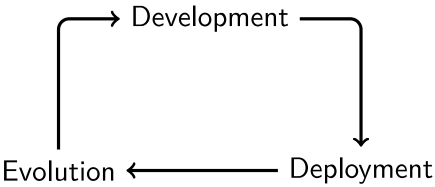
- platform = machine which understands DSL
- variation points = potential of DSL
- bind variation points = write specific DSL program
- binding time options
  - machine = interpreter
  - machine = code generator

Software Product Line Dimensions

Composition

- Architecture
- Components
- System

Life Cycle



Development

Deployment

Evolution

Software Product Line Dimensions

Views

- Business
- Architecture
- Process
- Organization

Advise

Key Success Factors

- Product Scoping
- Architectural Choice
- Level of Generalization
- Communication between Domain and Application Engineering

What have we seen today?

Terminology

- Software Product Line
- Commonality & Variability
- Feature Modelling

Examples

- motivation for software product lines
- commonality & variability analysis at different levels
- feature modelling for product lines
- architecture hints for product lines

Related Concepts

Related Concepts

- Software Architecture (OSGi, SOA, . . .)
- Model Driven Engineering
- Software Factories
- . . .

Challenges

- holistic approach
- manage variability in all artifacts
- find the right architecture
- . . .

Bibliography

Clements, Northrop:
Software Product Lines: Practices and Patterns
Addison-Wesley 2002.

Pohl, Böckle, van der Linden:
Software Product Line Engineering
Springer 2005.

Bosch:
Design & Use of Software Architectures
Pearson 2000.

Bibliography

- Czarnecki, Eisenecker:
  Generative Programming
  Addison-Wesley 2000.

- Greenfield, Short, Cook, Kent:
  Software Factories
  Wiley 2004.

- Kang, Cohen, Hess, Novak, Petersen:
  Feature-Oriented Domain Analysis
  SEI, Carnegie-Mellon University 1990.

- Parnas:
  On the Design and Development of Program Families. in:
  IEEE Transactions on Software Engineering, 2(1), 1976.