

## Part I: Ad-Hoc Approaches for Variability

1. Introduction
2. Runtime Variability and Design Patterns
3. Compile-Time Variability with Clone-and-Own

## Part II: Modeling & Implementing Features

4. Feature Modeling
5. Conditional Compilation
6. Modular Features
7. Languages for Features
8. Development Process

## Part III: Quality Assurance and Outlook

9. Feature Interactions
10. Product-Line Analyses
11. Product-Line Testing
12. Evolution and Maintenance

### 1a. Introduction to Product Lines

- Handcrafting and Customization
- Mass Production
- Mass Customization
- Recap: The Software Life Cycle
- Features and Products of a Domain
- Software Product Line
- Product-Line Engineering
- Summary

### 1b. Challenges of Product Lines

- Software Clones
- Feature Traceability
- Automated Generation
- Combinatorial Explosion
- Feature Interactions
- Continuing Change and Growth
- Summary

### 1c. Course Organization

- What You Should Know
- What You Will Learn
- What You Might Need
- Credit for the Slides
- Summary
- FAQ

# 1. Introduction – Handout

Software Product Lines | Thomas Thüm, Timo Kehrer, Elias Kuiter | April 19, 2023

# **1. Introduction**

## **1a. Introduction to Product Lines**

Handcrafting and Customization

Mass Production

Mass Customization

Recap: The Software Life Cycle

Features and Products of a Domain

Software Product Line

Product-Line Engineering

Summary

## **1b. Challenges of Product Lines**

## **1c. Course Organization**

# What do these examples have in common?



## Customization

- aka. handcrafting
- labor-intensive production
- highly individual goods

# Customization of Elevators



two buttons



one button



keyhole



floor display

# Customization of Elevators



no button to close door



two keyholes



keycard



double tap for undo

# Mass Production

## Mass Production

[Apel et al. 2013, pp. 3–4]

- consequence of industrialization
- goods are produced from standardized parts
- improved productivity wrt. handcrafting
- reduced costs, improved quality
- but: (almost) no individualism

## Principle: One Size Fits All

- t-shirts: XS, S, M, L, XL, XXL
- swiss-army knife



## Mass Production for Software?

[Apel et al. 2013, p. 7]

"The idea is to provide software that satisfies the needs of most customers, which leads almost automatically to the situation, in which customers miss desired functionality and are overwhelmed with functionality they do not need actually (just think of any contemporary office or graphics program). It is often this generality that makes software complex, slow, and buggy."

# About Every Second Car is Unique



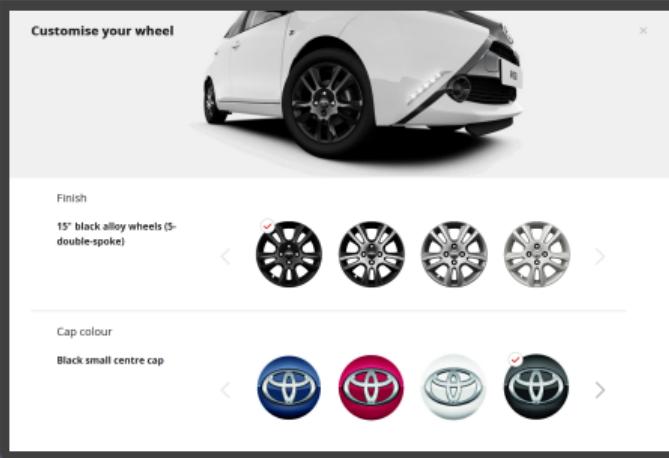
# Mass Customization

## Mass Customization

[Apel et al. 2013, p. 4]

- = mass production + customization
- customized, individual goods at costs similar to mass production

## Car Configuration



## Car Production



## Other Domains

bikes, computers, electronics, tools, medicine, clothing, food, financial services, . . . , software?

# Mass Customization for Software?

## Mass Customization for Software?

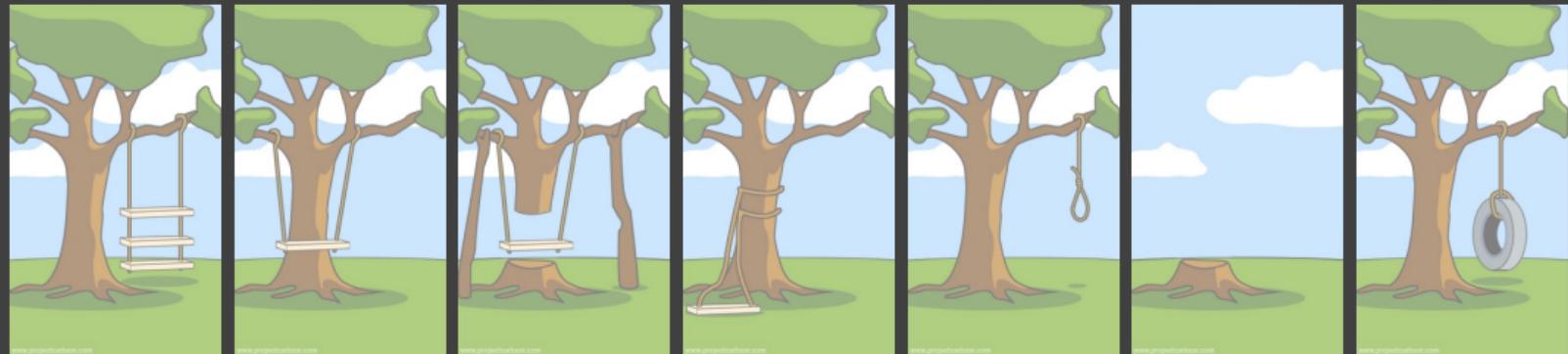
- customization: individual software developed using Waterfall model or Scrum
- mass production: standard software developed once for millions or billions of users (e.g., Whatsapp messenger)
- mass customization: software product lines

## Why Software Product Lines?

- resource limitations: memory, performance, energy
- different hardware
- different laws
- goal: avoid expensive customization
- how is software developed?



# The Project Cartoon



how the  
customer  
explained it

how the  
project leader  
understood it

how the  
analyst  
designed it

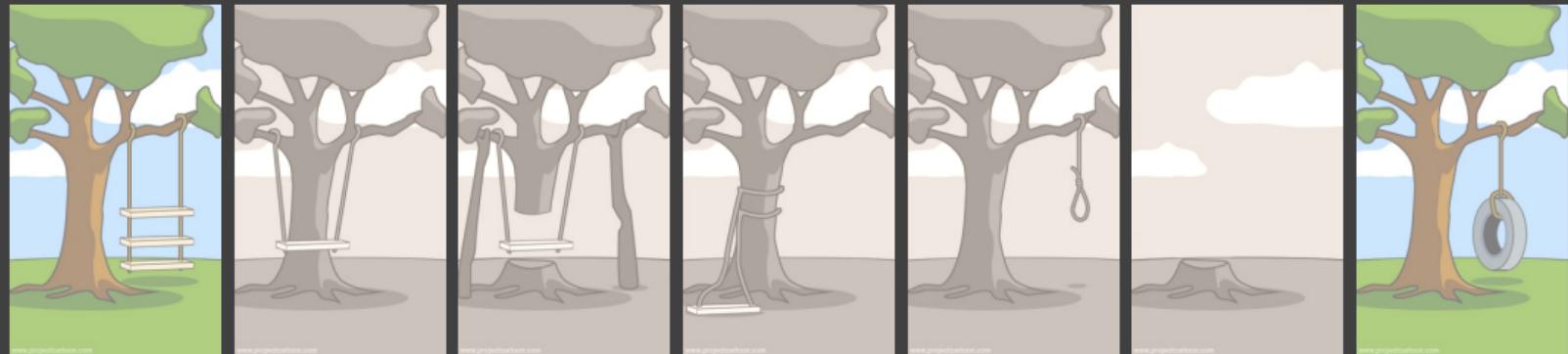
how the  
programmer  
implemented it

what the beta  
testers  
received

how it was  
supported

what the  
customer really  
needed

# The Project Cartoon



Requirements

how the  
project leader  
understood it

how the  
analyst  
designed it

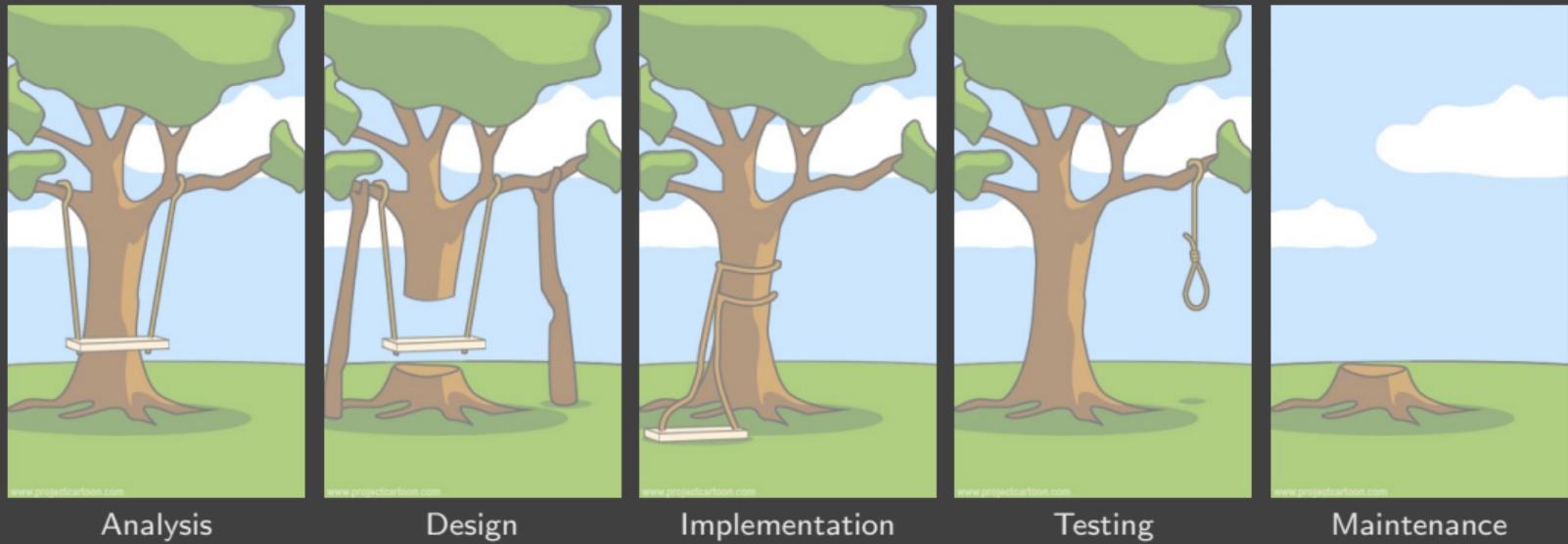
how the  
programmer  
implemented it

what the beta  
testers  
received

how it was  
supported

Product

# Recap: The Software Life Cycle

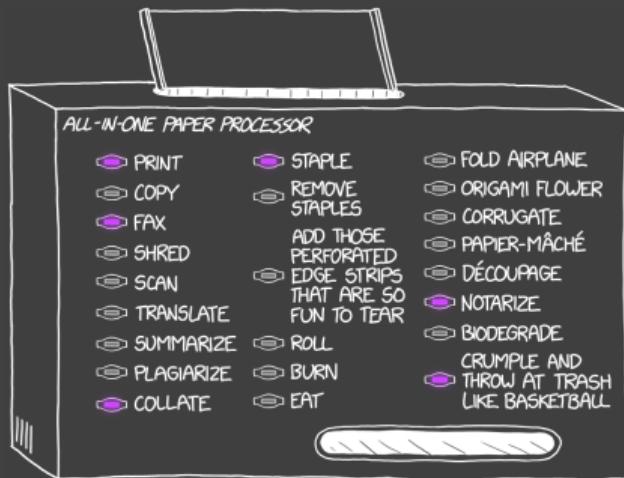


# What is a Feature?

## Feature

[Apel et al. 2013, p. 18]

"A **feature** is a characteristic or end-user-visible behavior of a software system."

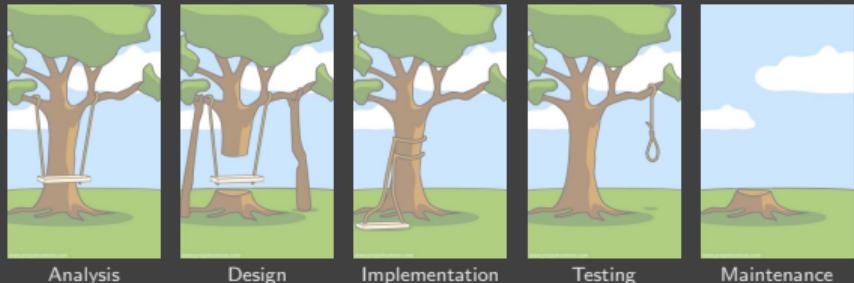


## Feature in a Product Line

[Apel et al. 2013, p. 18]

"Features are used in product-line engineering

- to specify and communicate commonalities and differences of the products between stakeholders and
- to guide structure, reuse, and variation across all phases of the software life cycle."



# What is a Product?

## Product

[Apel et al. 2013, p. 19]

"A **product of a product line** is specified by a valid feature selection (a subset of the features of the product line). A feature selection is **valid** if and only if it fulfills all feature dependencies."

## Note on Terminology

- in this course:  
product = product variant = variant
- software product: a product consisting only of software
- software is more than a program: requirements, models, source code, tests, documentation
- this course focuses on source code

## Product Map for Eclipse (excerpt)

	 Java	 Enterprise Java/Web	 C/C++	 Committees
C/C++ Development Tools				✓
Data Tools Platform			✓	
Git integration for Eclipse	✓	✓	✓	✓
Java Development Tools	✓	✓		✓
Java EE Developer Tools			✓	
JavaScript Development Tools				
Maven Integration for Eclipse	✓	✓		✓

# What is a Domain?

## Domain

[Apel et al. 2013, p. 19]

"A **domain** is an area of knowledge that:

- is scoped to maximize the satisfaction of the requirements of its stakeholders,
- includes a set of concepts and terminology understood by practitioners in that area,
- and includes the knowledge of how to build software systems (or parts of software systems) in that area."

## Features of a Domain

- a feature is a domain abstraction
- identification of features in a domain requires domain expertise
- later: select features for a product line?



# Software Product Line

## Software Product Line

[Northrop et al. 2012, p. 5]

"A **software product line** is

- a set of software-intensive systems

aka. products or variants

- that share a common, managed set of features

common set, but not all products have all features in common

- satisfying the specific needs of a particular market segment or mission

aka. domain

- and that are developed from a common set of core assets in a prescribed way."

aka. planned, structured reuse

[Software Engineering Institute, Carnegie Mellon University]

# Product-Line Engineering

## Product-Line Engineering

[Pohl et al. 2005, p. 14]

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

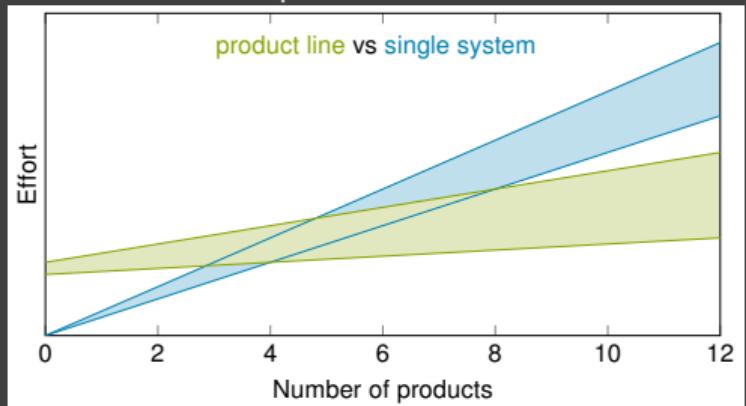
## Promises of Product Lines

[Apel et al. 2013, pp. 9–10]

- tailor-made
- reduced costs
- improved quality
- reduced time-to-market

## Idea of Product-Line Engineering

Reduce effort per product by means of an up-front investment for the product line:



## Single-System Engineering

classical software development that is not considered as product-line engineering

# Introduction to Product Lines – Summary

## Lessons Learned

- mass customization  
= mass production + customization
- features, products, domains
- software product lines
- product-line engineering

## Further Reading

- Apel et al. 2013, Chapter 1, pp. 3–15 — introduction close to this lecture, further examples
- Northrop et al. 2012, pp. 2–8 — what is not a product line?

## Practice

- What other examples of product lines do you know?
- Exemplify the differences between feature, product, domain, and product line for these examples.
- Are these product lines related to software?

# **1. Introduction**

## **1a. Introduction to Product Lines**

## **1b. Challenges of Product Lines**

Software Clones

Feature Traceability

Automated Generation

Combinatorial Explosion

Feature Interactions

Continuing Change and Growth

Summary

## **1c. Course Organization**

# Software Clones

## Software Clone

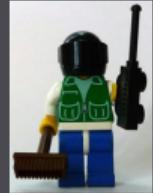
[Rattan et al. 2013, p. 1166]

- = result of copying and pasting existing fragments of the software
- code clones = copied code fragments
- replicates need to be altered consistently
- for example: bugs need to be fixed in all replicated fragments
- in practice: a common source for inconsistencies and bugs

## Cloning Parts of Software



## Cloning Whole Products (Clone-and-Own)



# Feature Traceability

## Feature Traceability

Feature traceability is the ability to trace a feature throughout the software life cycle (i.e., from requirements to source code).

## Intuition on Feature Traceability



find feature



in product

## Feature Traceability with Colored Source Code

A screenshot of a Java IDE interface titled "FeatureDE - FeatureDE - Elevator-Antenna-v1.4/src/de/ovgu/featureide/examples/elevator/core/controller/Request.java...". The code editor displays a Java file named "Request.java". The code is color-coded to show feature traceability:

```
// @return (floor != other.floor)
// #endif
public static class RequestComparator implements Comparator<Request> {
    // #if ShortestPath
    protected ControlUnit controller;
    public RequestComparator(ControlUnit controller) {
        this.controller = controller;
    }
    // #endif
    @Override
    public int compare(Request o1, Request o2) {
        // #if DirectedGraph
        return compareDirectional(o1, o2);
        // #else
        // #if FIFO
        // @return (int) Math.signum(o1.timestamp - o2.timestamp)
        // #endif
        // #if ShortestPath
        // @int diff0 = Math.abs(o1.floor - controller.getControlUnit().getFloor());
        // @int diff1 = Math.abs(o2.floor - controller.getControlUnit().getFloor());
        // @return diff0 - diff1;
        // #endif
        // #endif
    }
}
```

The code editor has a toolbar at the top and a sidebar on the right containing various icons and buttons. The status bar at the bottom shows "Writable", "Smart Insert", and "1:1".

# Automated Generation

Features



Products



# Automated Generation

## Product Line with Features



### Goal

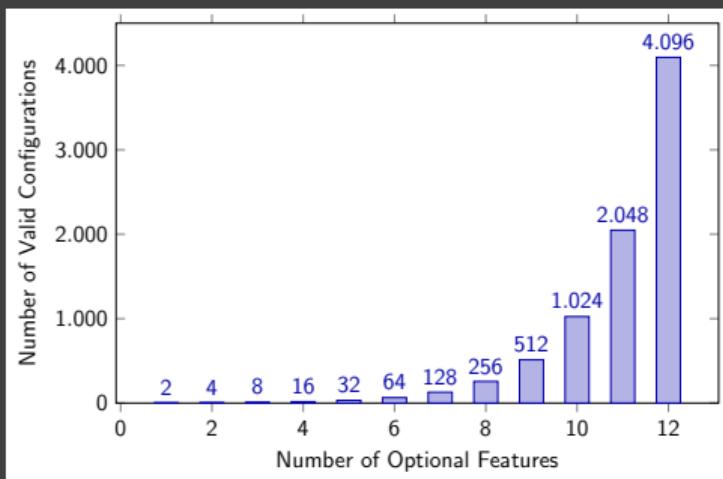
- automatic generation of products
- based on a (descriptive) selection of features

### Challenges

- how to map features to source code?
- how to combine source code of multiple features?
- how to define valid combinations of features?

# Combinatorial Explosion

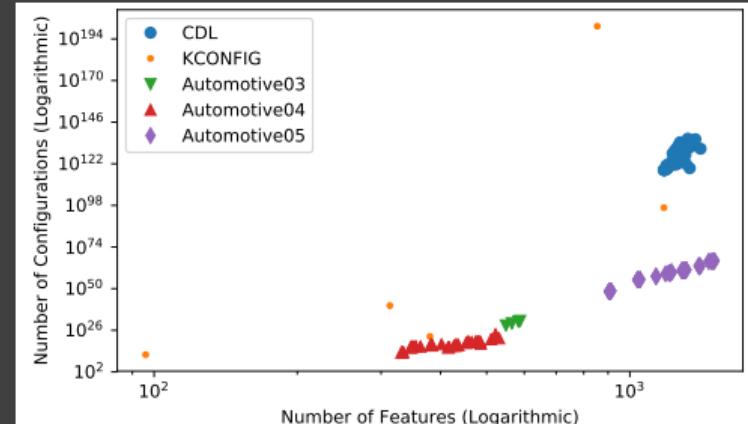
## Combinatorial Explosion



- assumption: all combinations of features are valid
- 33 features: a unique combination for every human
- 320 features: more combinations than atoms in the universe

## Industrial Configuration Spaces

[Sundermann et al. 2020]

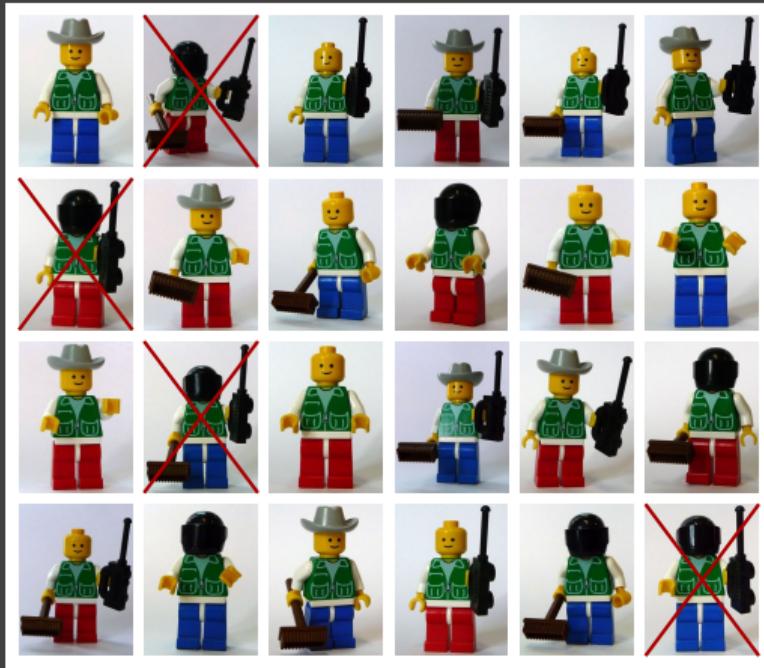


- in practice: not all combinations of features valid
- many industrial product lines too large to specify all valid combinations separately
- largest automotive product line has about  $1.7 \cdot 10^{1534}$  products

# Combinatorial Explosion



# Feature Interactions



## Example Interaction



phone  cannot be used with helmet



## Challenges

- interaction typically unknown in advance
- interactions occur in some but not all combinations
- challenge for quality assurance

# Feature Interactions

## Invalid Car Configurations

Configuration Assistant.

› Show instructions

Your most recent action requires your configuration to be adjusted.

Your choice	Price
+ Enhanced Bluetooth telephone with USB & Voice Control	+ £ 350.00
Adding	
+ BMW Navigation	£ 0.00
Removing	
- Enhanced Bluetooth with wireless charging	- £ 395.00
- Navigation system Professional	£ 0.00
- WiFi hotspot preparation	£ 0.00
- Media package - Professional	- £ 900.00
- Online Entertainment	£ 0.00
- Microsoft Office 365	- £ 150.00

# Continuing Change and Growth

## Lehman's Laws of Software Evolution (excerpt)

[Lehman et al. 1997]

- Continuing Change: systems must be continually adapted to stay satisfactory
- Increasing Complexity: complexity increases during evolution unless work is done to maintain or reduce it
- Continuing Growth: functionality must be continually increased to maintain user satisfaction
- Declining Quality: quality will decline unless rigorously maintained and adapted to operational environment changes

## Essence of the Laws

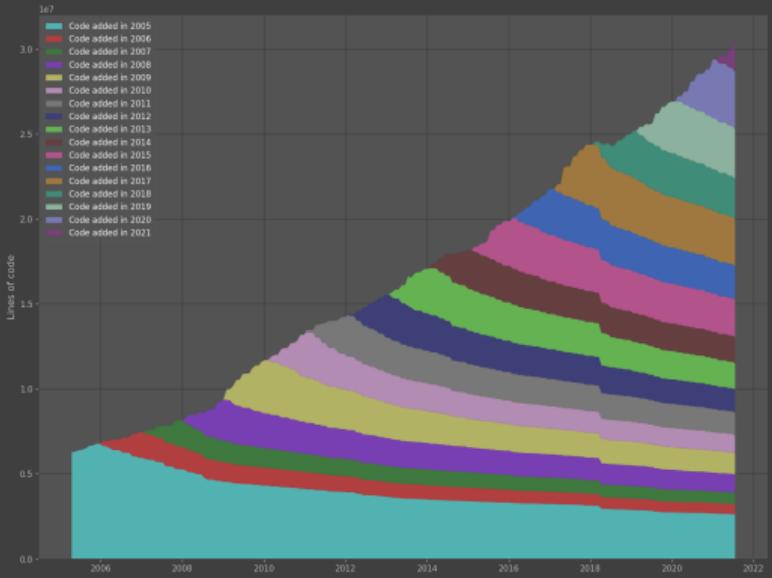
- software that is used will be modified
- when modified, its complexity will increase (unless one does actively work against it)

## Consequences for Product Lines

- number of features and size of implementation increases over time
- discussed challenges increase over time
  - more software clones
  - harder to trace features
  - automated generation more urgent
  - increasing combinatorial explosion
  - more feature interactions

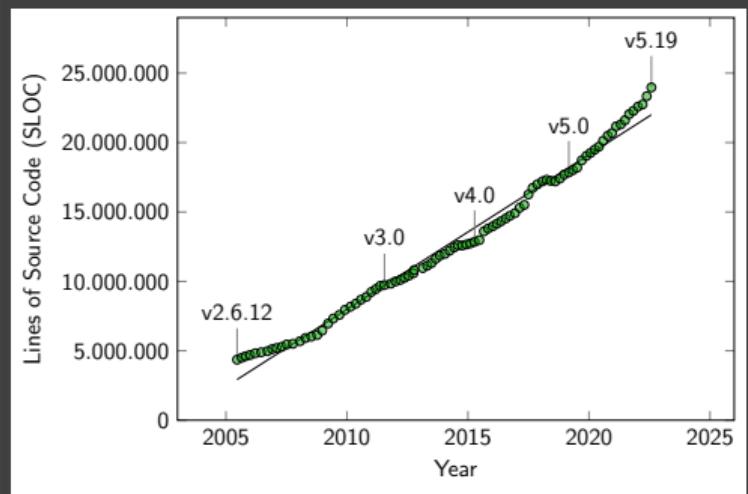
# Evolution of the Linux Kernel

- about 60,000 commits per year
- in peak weeks: new commit every 5 minutes
- in average weeks: every 9 minutes



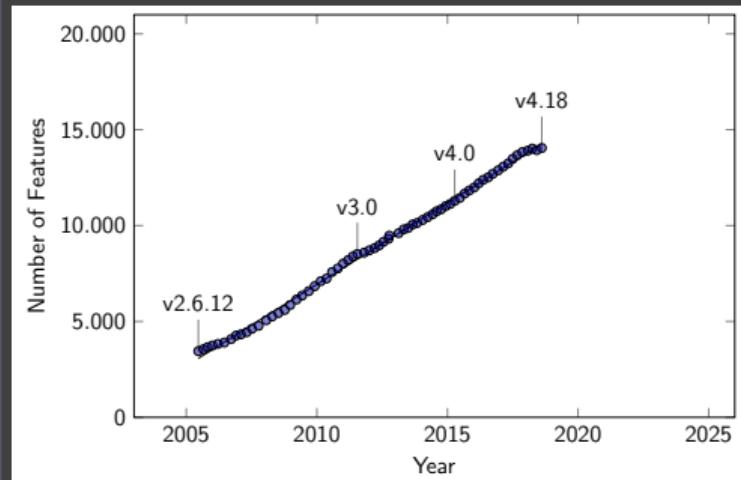
# Evolution of the Linux Kernel

Size of the Code Base



- from 4 to 24 millions in 17 years
- about one million LOC added every year
- about 3,000 LOC per day

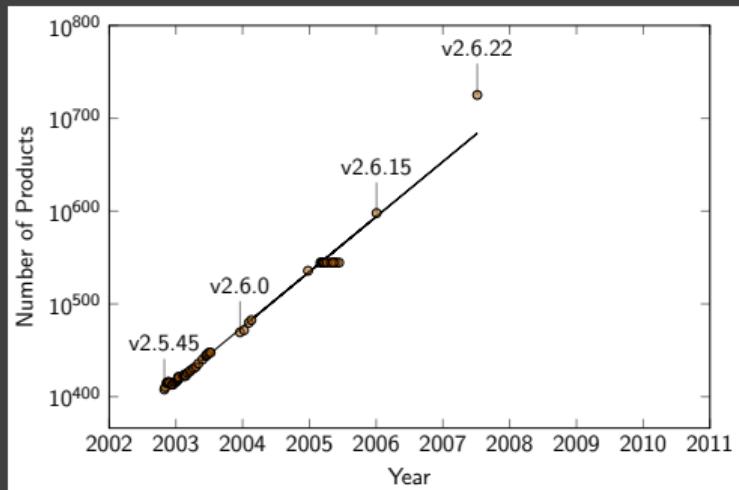
Number of Features



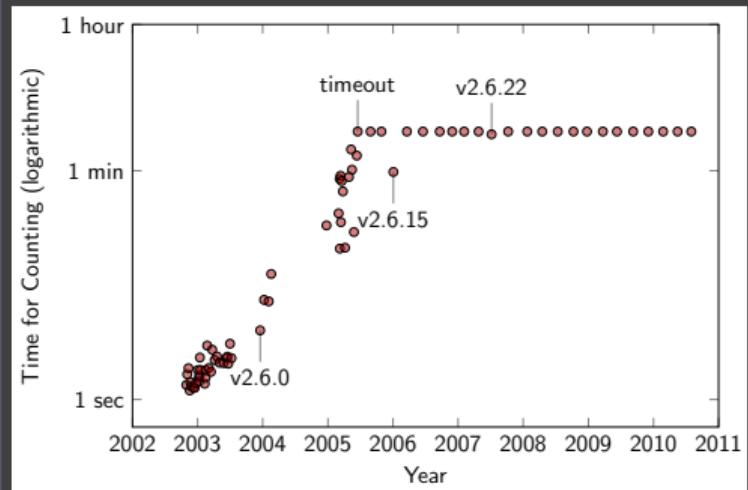
- about 800 new features every year
- about 15 new features every week
- in 2018 four times more features than in 2005

# Evolution of the Linux Kernel

Number of Products



Time to Count Products



- number of products grows by factor 100.000 each month
- the current kernel is likely to have more than  $10^{1500}$  products

- most kernel versions before 2006 can be computed within 1 minute
- most kernel versions after 2006 cannot be computed within 1 hour

# Challenges of Product Lines – Summary

## Lessons Learned

- why are product lines challenging?
- selected challenges:
  1. software clones
  2. feature traceability
  3. automated generation
  4. combinatorial explosion
  5. feature interactions
  6. continuous growth

## Further Reading

see later lectures

## Practice

- Form groups of 2–3 students
- Explain 2–3 of the six challenges to your colleagues
- Can you find own examples for these challenges?

# **1. Introduction**

## **1a. Introduction to Product Lines**

## **1b. Challenges of Product Lines**

## **1c. Course Organization**

What You Should Know

What You Will Learn

What You Might Need

Credit for the Slides

Summary

FAQ

# What You Should Know

## Fundamentals of Software Engineering

- development processes
- object-oriented programming
- design patterns
- UML class diagrams
- modularity

## Fundamentals of Theoretical Computer Science

- set theory
- propositional logic
- complexity theory

## Exercise

solid programming skills in Java

# What You Will Learn

## Part I: Ad-Hoc Approaches for Variability

1. Introduction
2. Runtime Variability and Design Patterns
3. Compile-Time Variability with Clone-and-Own

## Part II: Modeling & Implementing Features

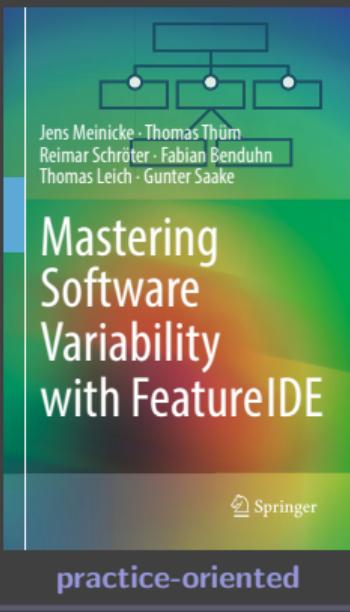
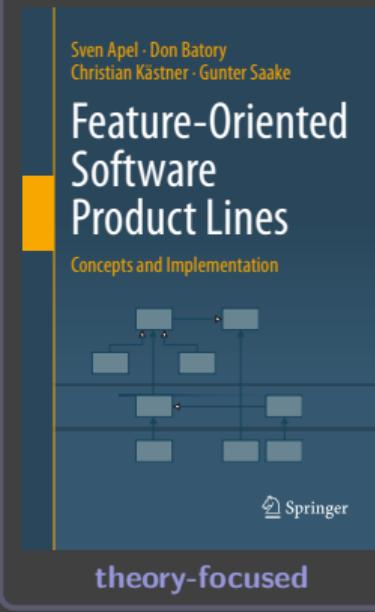
4. Feature Modeling
5. Conditional Compilation
6. Modular Features
7. Languages for Features
8. Development Process

## Part III: Quality Assurance and Outlook

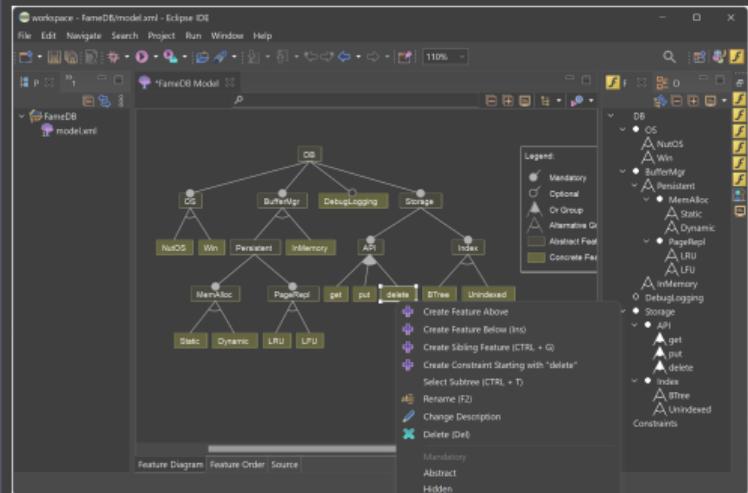
9. Feature Interactions
10. Product-Line Analyses
11. Product-Line Testing
12. Evolution and Maintenance

# What You Might Need

## Recommended Literature for Lecture & Exercise



## Recommended Tool Support for the Exercise



# Credit for the Slides



Thomas Thüm



Professor at TU Braunschweig  
software engineering  
FeatureIDE team leader

Timo Kehrer



Professor at University of Bern  
software engineering

Elias Kuiter



PhD student in Magdeburg  
feature-model analysis  
FeatureIDE core developer

# Course Organization – Summary

## Lessons Learned

- focus: how to implement features
- focus: how to model valid combinations
- focus: how to do quality assurance
- course organization

## Further Reading

- Apel et al. 2013 — best book for this lecture
- Meinicke et al. 2017 — more practical guide on tool support

## Practice

- Ask questions on the course organization!
- Form teams for the practical tasks

# FAQ – 1. Introduction

## Lecture 1a

- What is a software product line, feature, product/variant, domain?
- What is customization, handcrafting, mass production, mass customization?
- What are example for each of those?
- What is the one-size-fits-all or swiss-army-knife principle?
- What is the difference between product-line engineering and single-system engineering?
- What are advantages of product-line engineering?

## Lecture 1b

- What are software clones, feature traceability, automated generation, combinatorial explosion, feature interactions, and continuous growth?
- Why are those challenging when developing product lines?
- What are examples for these six fundamental challenges?
- How do those challenges interact with each other?
- How complex is the Linux kernel?
- At which pace is the Linux kernel developed?

## Lecture 1c

- What you should know?
- What you will learn?
- What you might need?
- Who are the authors of this course?
- How is this course organized?