Software
Systems
Engineering

Towards identifying evolution smells in Software Product Lines

Klaus Schmid[1], Rainer Koschke[2],
Christian Kröher[1], Dierk Lüdemann[2]

[1]University of Hildesheim,
Institute of Computer Science,
Software Systems Engineering

Marienburger Platz 22,
D-31141 Hildesheim, Germany

{schmid, kroeher}@sse.uni-hildesheim.de

[2]University of Bremen,
Institute of Computer Science,
Softwaretechnik

Am Fallturm 1,
D-28359 Bremen, Germany

koschke@informatik.uni-bremen.de
dierk@tzi.de

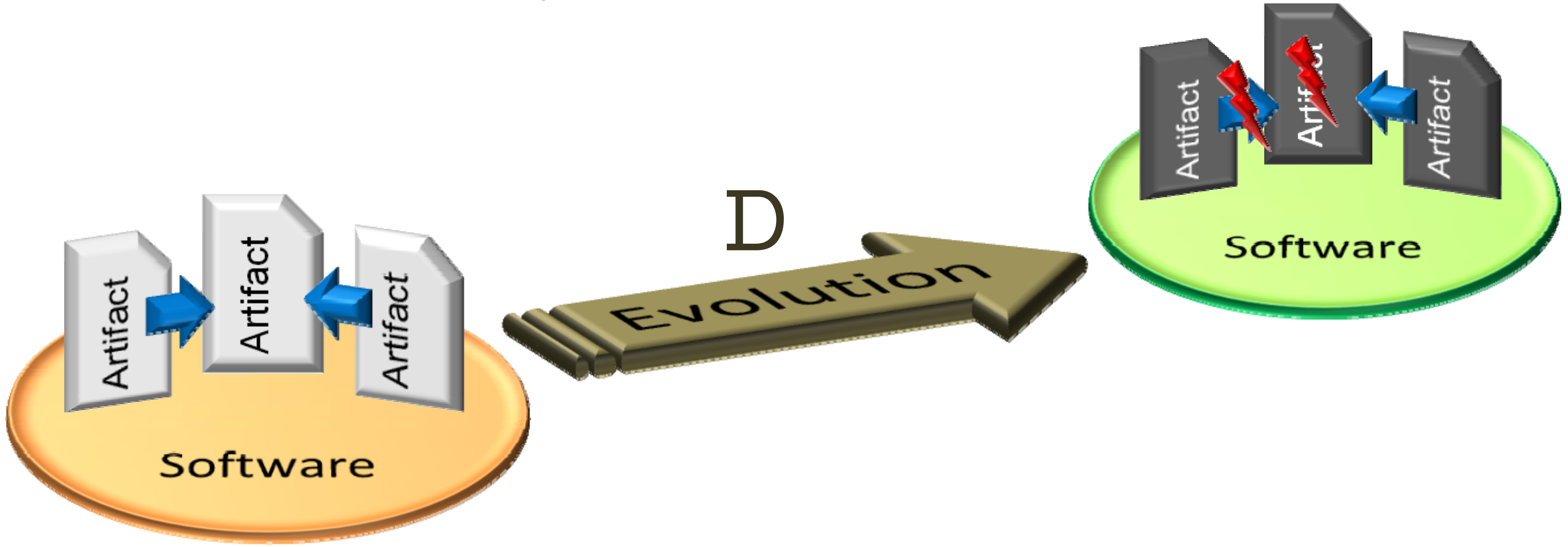
Contents

Towards identifying evolution smells in Software Product Lines

Motivation

What is the problem of software product line evolution?

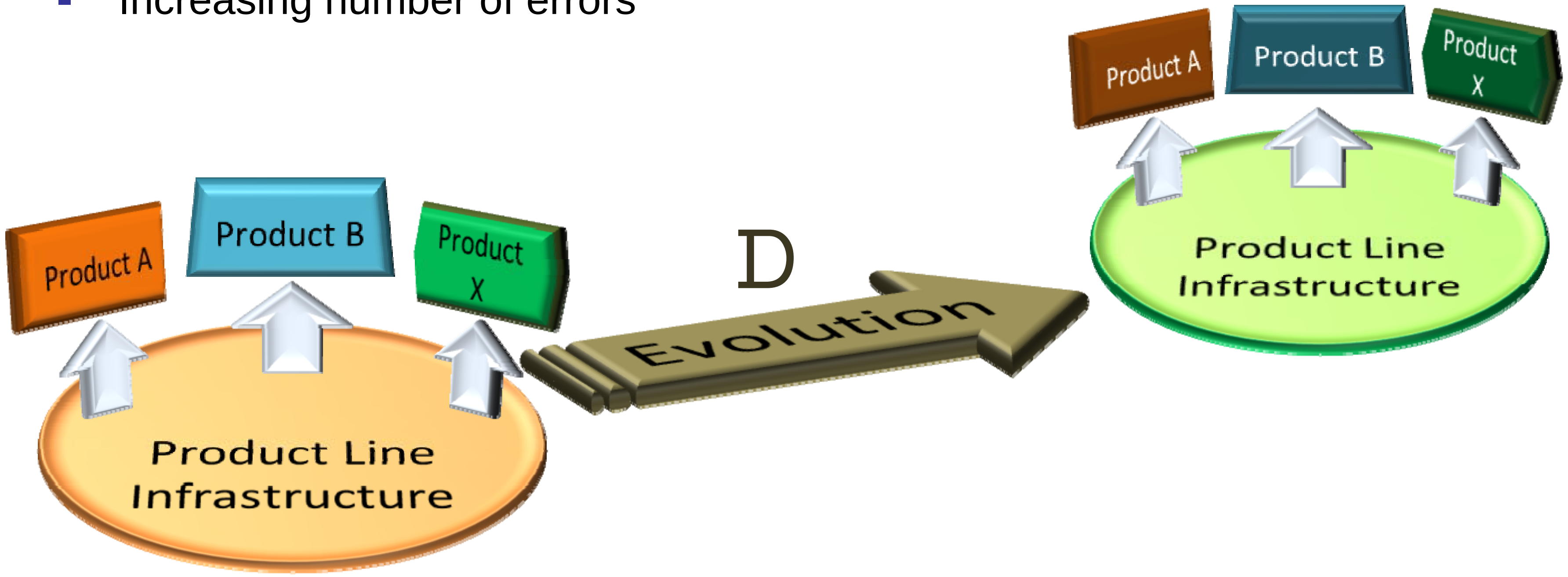
- Dependencies among parts of the software
- Introduction of errors



Motivation

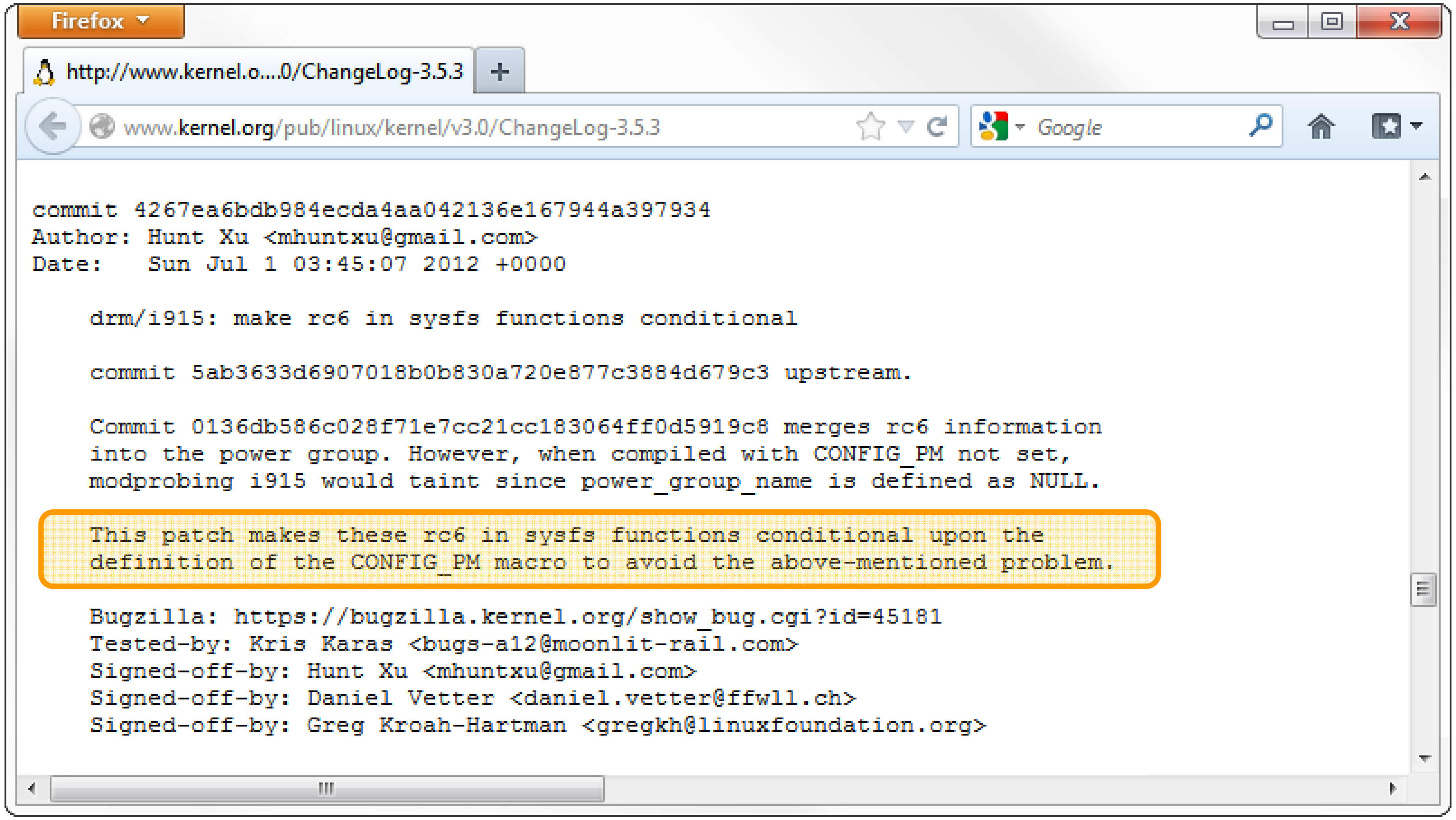
What is the problem of software product line evolution?

- Increasing complexity in software product line situations
- Increasing number of errors



Motivation

Some observations from the real world

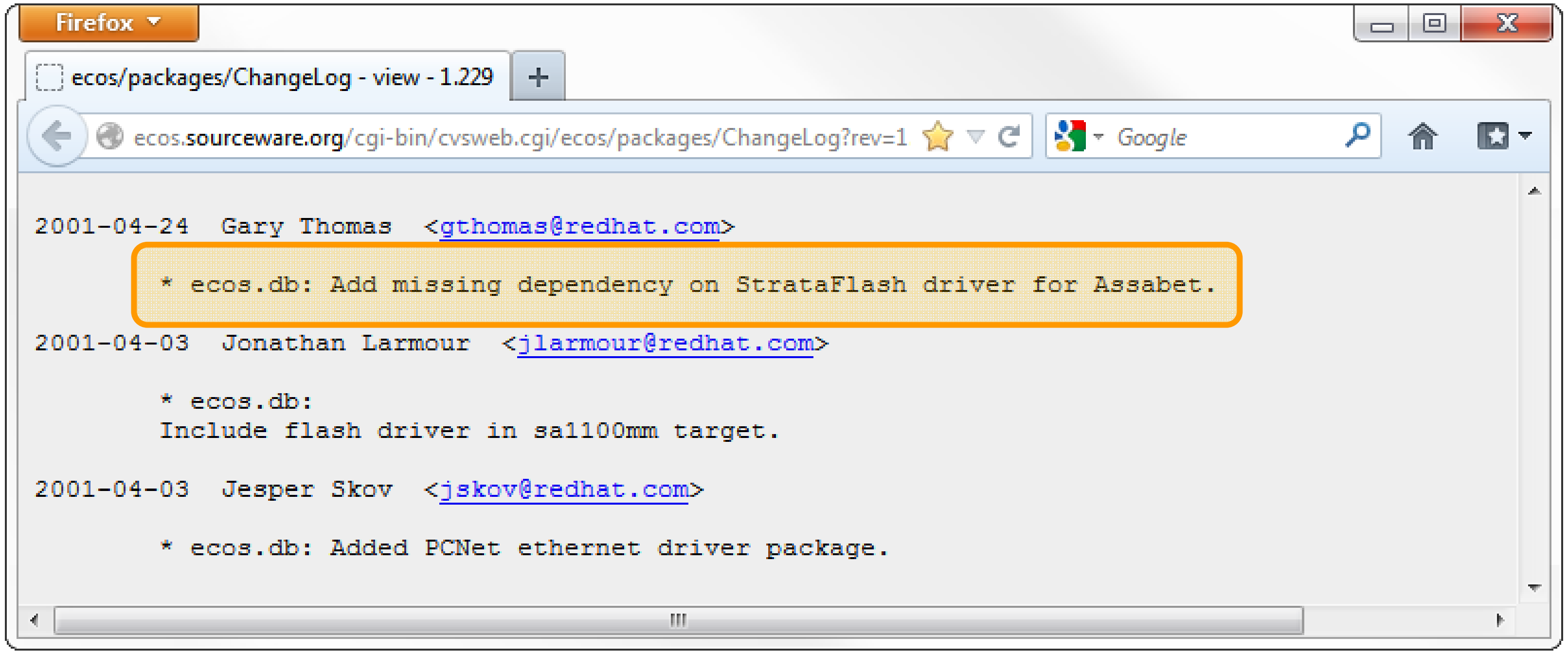

Firefox
http://www.kernel.o....0/ChangeLog-3.5.3
www.kernel.org/pub/linux/kernel/v3.0/ChangeLog-3.5.3
Google

```
commit 4267ea6bdb984ecda4aa042136e167944a397934
Author: Hunt Xu <mhuntxu@gmail.com>
Date:   Sun Jul 1 03:45:07 2012 +0000

    drm/i915: make rc6 in sysfs functions conditional

    commit 5ab3633d6907018b0b830a720e877c3884d679c3 upstream.

    Commit 0136db586c028f71e7cc21cc183064ff0d5919c8 merges rc6 information
    into the power group. However, when compiled with CONFIG_PM not set,
    modprobing i915 would taint since power_group_name is defined as NULL.

    This patch makes these rc6 in sysfs functions conditional upon the
    definition of the CONFIG_PM macro to avoid the above-mentioned problem.

    Bugzilla: https://bugzilla.kernel.org/show_bug.cgi?id=45181
    Tested-by: Kris Karas <bugs-a12@moonlit-rail.com>
    Signed-off-by: Hunt Xu <mhuntxu@gmail.com>
    Signed-off-by: Daniel Vetter <daniel.vetter@ffwll.ch>
    Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
```

Towards identifying evolution smells in Software Product Lines

Motivation

Some observations from the real world


Firefox
ecos/packages/ChangeLog - view - 1.229
ecos.sourceware.org/cgi-bin/cvsweb.cgi/ecos/packages/ChangeLog?rev=1
Google

```
2001-04-24  Gary Thomas  <gthomas@redhat.com>

        * ecos.db: Add missing dependency on StrataFlash driver for Assabet.

2001-04-03  Jonathan Larmour  <jlarmour@redhat.com>

        * ecos.db:
        Include flash driver in sa1100mm target.

2001-04-03  Jesper Skov  <jskov@redhat.com>

        * ecos.db: Added PCNet ethernet driver package.
```

Motivation

Reasons for increasing complexity in SPL-situations (1/2)

1. Longevity
   – Product lines are mayor investments for companies
   – Product lines evolve as long as their products evolve
2. Impact of modifications
   – Correction for one product = Defect for another product
   – Range of products, but **not all** products
   – Unclear impact on individual products

Motivation

Reasons for increasing complexity in SPL-situations (2/2)

3. Large artifact space
   - More artifacts (e.g., infrastructure, variable part)
   - Larger size of artifacts (e.g., due to the inclusion of variants)
4. Complex relations
   - Existence of variability models
   - Variability in models vs. variability in artifacts

Towards identifying evolution smells in Software Product Lines

Motivation

The EvoLine project

- Ways to identify problems that are introduced as part of product line evolution

➔ **evolution smells**

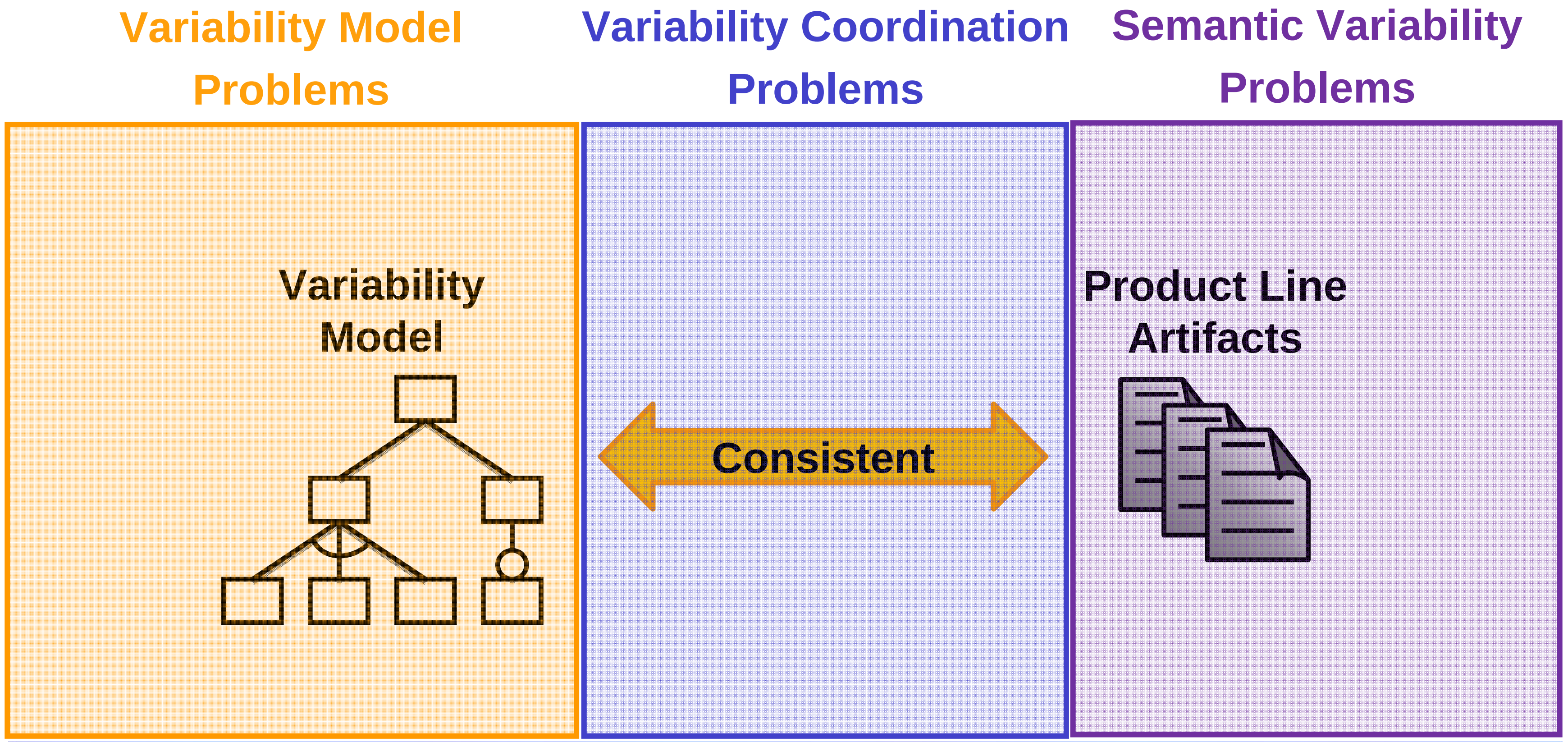
Evolution smells are deltas that seem to introduce problems

- Focus:
  – (Semi)-automatic detection of problems that may exists or be introduced in product lines
  – **Evolutionary analysis**
- Assumptions:
  – A (consistent) product line is already established
  – No discussion of the problem of identifying, e.g., copy-and-paste reuse

Part of the German Research Foundation (DFG)
Priority Programme SPP 1593

Approaches to Analyze Defects

Three mayor directions relevant to analyzing quality problems in product lines


Variability Model Problems
Variability Coordination Problems
Semantic Variability Problems
Variability Model
Consistent
Product Line Artifacts

Approaches to Analyze Defects

Variability model problems

- Analyze the variability model for any obvious problems
- Examples:
  - Analyze inconsistencies in the variability model
  - Dead feature analysis
- Key characteristics:
  - Analysis of the variability model only

Approaches to Analyze Defects

Semantic variability problems

- Analyze semantic information from the code more deeply
- Examples:
  - Type inconsistencies
  - Data-flow issues
- Key characteristics:
  - Analysis of the variability realization only

Approaches to Analyze Defects

Variability coordination problems

- Identification of inconsistencies between variability model and realization
- But: Which one is correct?
- Example:
  - C-Preprocessor (prime interest in EvoLine)
  - Two features A and B
  - *Implied* dependency: B → A
- Key characteristics:
  - Analysis of the relations between variability model and realization only

Approaches to Analyze Defects

State of the art

- Many examples of these kinds of analyses exist
- Individual analyses of specific aspects (e.g., variability model only)
- Focus on a specific state of the software / product line
- Analyses are rather inefficient with respect to large-scale product lines

Why EvoLine? - Beyond State of the Art

- Focus on evolution
- Combination of the results of different analyses
- Identification of problems that could not be identified previously
  ➔ evolution smells
- Increase of efficiency of analysis methods

Evolutionary Analysis

The approach

- Focus on the change (the delta between two states of a product line)
  - Assumption: Product line was correct before
  - Only interest is the change itself
- Slight perspective change:
  - Correct product line PL
  - Change C
  - Evolved product line $\mathrm{PL'}$

Instead of asking $\mathit{correct}(\mathrm{PL'})$, we ask $\mathit{correct}(\mathrm{PL}) \rightarrow \mathit{correct}(\mathrm{PL'})$

Evolutionary Analysis

Expected benefits

- Reduction of information
  - Individual change is typically smaller than the product line
  - Interpretation of change in context
- Reduction of complexity
- Faster identification of (possible) problems

Conclusion

- Introduction of the concept of *product line evolution smells*
- Introduction of the *evolutionary analysis* approach
  - Optimization of existing analyses through incremental semantics
  - Provision of faster results to the developers (making the analysis more useful)
  - Focus on those parts that have recently changed (making the results more relevant)

➔ Basis for research in progress

➔ Studied further in the EvoLine-project

Towards identifying evolution smells in Software Product Lines

Thank you for your attention.
Questions?