Towards identifying evolution smells in Software Product Lines

Klaus Schmid¹, Rainer Koschke², Christian Kröher¹, Dierk Lüdemann²

¹University of Hildesheim, Institute of Computer Science, Software Systems Engineering
Marienburger Platz 22, D-31141 Hildesheim, Germany
{schmid, kroehrer}@sse.uni-hildesheim.de

²University of Bremen, Institute of Computer Science, Softwaretechnik
Am Fallturm 1, D-28359 Bremen, Germany
koschke@informatik.uni-bremen.de
dierk@tzi.de
Towards identifying evolution smells in Software Product Lines

Contents

1. Motivation
2. Approaches to Analyze Defects
3. Evolutionary Analysis
4. Conclusion
What is the problem of software product line evolution?

- Dependencies among parts of the software
- Introduction of errors
Towards identifying evolution smells in Software Product Lines

Motivation

What is the problem of software product line evolution?

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

Product A  Product B  Product X

Product Line Infrastructure

Product Line Infrastructure
Towards identifying evolution smells in Software Product Lines

Motivation

Some observations from the real world

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

commit 4267ea6b6b984ecda1aaa042136ae167944a397934
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 0136db536c028f71e7c21cc183064ff0d5919c8 merges rc6 information into the power group. However, when compiled with CONFIG_PM not set, modprobing i918 would taint since power_group_name is defined as NULL.

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

- 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 sal100mm 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 major 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
Motivation

The EvoLine project

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

Focus:
- (Semi)-automatic detection of problems that may exist 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

*Evolution smells are deltas that seem to introduce problems*
Towards identifying evolution smells in Software Product Lines

Approaches to Analyze Defects

Three major directions relevant to analyzing quality problems in product lines

- Variability Model Problems
- Variability Coordination Problems
- Semantic Variability Problems
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
Towards identifying evolution smells in Software Product Lines

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
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
Towards identifying evolution smells in Software Product Lines

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
Towards identifying evolution smells in Software Product Lines

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 PL’

Instead of asking \textit{correct}(PL’), we ask \textit{correct}(PL) \rightarrow \textit{correct}(PL’)

01.03.2013, DFF 2013 © SSE, University of Hildesheim
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
Thank you for your attention. Questions?