Analyzing similarity of multiple cloned software systems

Slawomir Duszynski

slawomir.duszynski@iese.fraunhofer.de

Fraunhofer IESE

Kaiserslautern, Germany

November 28, 2011
The 16th CREST Open Workshop
UCL London
Motivation for Multi-System Analysis

- The need for systematic software reuse is often recognized only after development of a group of similar software systems
  - Common practice: clone and adapt one of existing variants, no reuse mechanisms
  - “Software mitosis” (Faust 2003)
  - Variants are maintained independently from each other
  - Further variants emerge in the same way

- Examples from the industry
  - 4 cloned variants, ca. 1.5 MLOC each
  - 14 cloned variants, ca. 200 KLOC each

- With a growing number of variants, maintenance becomes difficult
  - Redundant maintenance and QA effort

[D. Faust, C. Verhoef: Software Product Line Migration and Deployment. 2003]
[D. Beuche: Transforming Legacy Systems into Software Product Lines. SPLC 2010]
Motivation for Multi-System Analysis

- Having many similar variants, the company has two options:
  - **1: Develop** a new PL from scratch – costly, loss of past investment
  - **2: Migrate** the existing products – difficult, and costly too

- Typical migration problems
  - Variability in the existing code is not known
  - Code-level variability might differ from feature-level variability (Yoshimura 2006a)
  - High risk of incorrect reuse decisions (Garlan 1995; Kolb 2006)

- Research problem: detailed information about the code variability is needed
  - variability needs to be recovered and understood
  - difficult for large systems and many variants


“the portion of functional commonality among two products is about 60-75%; their implementations, however, share as little as around 30% of code”
We need an analysis technique that:

- Provides both abstract and detailed information
  - Available for any part of the code
  - Available for any variant or variant intersection

- Is scalable
  - High number of LOC
  - High number of variants
  - Suitable abstraction needed (providing just a flat list of similarities is not scalable!)

- Is specifically targeted at variants, not versions
  - Versions form a time-ordered list
    - It is enough to analyze n-1 pairs
  - Variants exist in parallel and cannot be ordered
    - Analysis of $\frac{n(n-1)}{2}$ pairs needed
    - Result cannot depend on any variant ordering

- [IESE context] Is understandable to practitioners
Existing Approaches

- Similarity metrics calculated on the whole systems (Yamamoto2005)
  - Only high-level information: it is known that there are differences, but it is not known where they are

- Clone detection and manual result analysis (Yoshimura2006b)
  - No scalability (lots of manual work, for just 2 variants)

- Clone detection and further result processing (Mende2008)
  - Unsuitable result presentation

[T. Yamamoto, M. Matsushita, T. Kamiya, K. Inoue: Measuring similarity of large software systems based on source code correspondence. 2005]
[K. Yoshimura, D. Ganesan, and D. Muthig: Defining a strategy to introduce a software product line using existing embedded systems. EMSOFT 2006]
Existing Approaches

Information on Any Variant Intersection: Not Available

**Problem**
- **Problem**: incomplete information
- **Example 1**: Two different situations (above) cannot be distinguished as they provide the same pair-wise result.
  - Example 2: impossible to answer questions such as “where is the core of my potential product line?”

**Problem**
- **Problem**: complex result
  - O(n²) variant pairs!

---

**Pair-wise result presentation**

<table>
<thead>
<tr>
<th></th>
<th>FW</th>
<th>( PP^1 )</th>
<th>FW</th>
<th>( P^1 )</th>
<th>( P^2 )</th>
<th>( P^3 )</th>
<th>( P^4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW</td>
<td>5</td>
<td>(0%)</td>
<td>1</td>
<td>(0%)</td>
<td>3</td>
<td>(0%)</td>
<td>1</td>
</tr>
<tr>
<td>( PP^1 )</td>
<td>1</td>
<td>(0%)</td>
<td>11</td>
<td>(90%)</td>
<td>117</td>
<td>(11%)</td>
<td>178</td>
</tr>
<tr>
<td>( P^1 )</td>
<td>3</td>
<td>(0%)</td>
<td>162</td>
<td>(93%)</td>
<td>-</td>
<td></td>
<td>162</td>
</tr>
<tr>
<td>( P^2 )</td>
<td>1</td>
<td>(0%)</td>
<td>186</td>
<td>(93%)</td>
<td>117</td>
<td>(93%)</td>
<td>2</td>
</tr>
<tr>
<td>( P^3 )</td>
<td>11</td>
<td>(99%)</td>
<td>57</td>
<td>(1%)</td>
<td>23</td>
<td>(1%)</td>
<td>0</td>
</tr>
<tr>
<td>( P^4 )</td>
<td>9</td>
<td>(0%)</td>
<td>89</td>
<td>(1%)</td>
<td>27</td>
<td>(1%)</td>
<td>20</td>
</tr>
</tbody>
</table>

Result presentation in (Mende2008)
Variant Analysis
Example Situation

- Consider three source code files A, B and C
  - The task: recognize and characterize the commonalities and variabilities
  - A human could use the diff tool to understand the differences

- Practical problems in a product line context:
  - Scalability problem: for n systems there are $\frac{n(n-1)}{2}$ pairs. Hard to understand for a human (e.g. n=6 $\rightarrow$ 15 different pairs to be related to each other)
  - Comparison delivers pair-wise results such as “same” and “different”: but for the product line, we want to know which lines are core and which are unique
Variant Analysis
Occurrence Matrices

- For each variant, list its elements in a matrix
- Add union matrix to represent the total analyzed code
- Fill the matrix
  - Rows: variant elements
  - Columns: all the existing variants; additionally: number of variants where the element occurs
  - Cells: occurrence of the elements in the variants (1: occurrence, 0: no occurrence)

- Redefine the line status to make it appropriate for product lines
  - Not “same” and “different”, but “core” (Sum=n), “shared”, “unique” (Sum=1)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>A3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>A4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>A5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>A6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>A7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>A9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>B1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Variant Analysis
n-ary Diff Results

- Instead of a group of diff-ed pairs...

- the result is a n-ary diff performed on all the involved variants:

- Using the same principle, a comparison for any number of variants is possible
Variant Analysis – Visualization

Venn Diagrams: Not the way to go…

- Venn diagrams: very useful for small number of sets

![Venn Diagrams](image)

- Harder to understand for larger number of sets

Number of diagram areas = $2^n$
Variant Analysis
Visualization: Bar Diagrams

- Bar diagrams are a way to visualize occurrence matrices
  - One bar created for each occurrence matrix (in total: n+1 bars)

- Size of the bar = number of elements in the matrix
- Bar parts symbolize the core, shared and unique elements in the variants
- Sizes of the particular parts reflected in the diagram
The information provided by Variant Analysis is complete
- Two example situations easily distinguishable
- Any set intersection can be obtained using subset calculations
- It is known how many elements fulfill a criterion and which elements they are
- Information can be easily presented even for a high number of variants
Variant Analysis
Subset Calculations

- Sometimes a specific subset of the analyzed system group is interesting, e.g.:
  - All elements shared by at least k systems
  - Elements common for a given system and other systems
  - Subsets such as $A \cap \neg B \cap \neg C \cap D$

- Subset elements can be found by evaluating the element occurrences in the matrix
- Visualization on a bar diagram: display relevant bar parts and associated numbers
- Visualization in text editor: highlight relevant text lines in the text editor
Variant Analysis

Scalable Result Abstraction and Navigation

- Variant Analysis integrated into Fraunhofer SAVE tool (Eclipse plug-in)
- Top-down result exploration possible using structural architectural views
  - Detect interesting areas on the high level structure
  - Go to details only where relevant results exist
- Example: the folders “core” vs. “data” in the figure
### Variant Analysis

**Industrial Application**

- **Good scalability and performance**
  - Four 1.5 MLOC variants (implemented in C++) analyzed in **7 minutes**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>663618</td>
<td>47</td>
<td>356371</td>
<td>1122110</td>
</tr>
<tr>
<td>B</td>
<td>663618</td>
<td>42</td>
<td>581592</td>
<td>171573</td>
</tr>
<tr>
<td>C</td>
<td>663618</td>
<td>41</td>
<td>715238</td>
<td>104</td>
</tr>
<tr>
<td>D</td>
<td>663618</td>
<td>41</td>
<td>292059</td>
<td>193582</td>
</tr>
<tr>
<td>ALL</td>
<td>663618</td>
<td>50</td>
<td>972630</td>
<td>524926</td>
</tr>
</tbody>
</table>

- Subset calculations on all rows time range **from 312ms to 328ms**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>663618</td>
<td>47</td>
<td>158572</td>
<td>237799</td>
</tr>
<tr>
<td>B</td>
<td>663618</td>
<td>42</td>
<td>561999</td>
<td>171573</td>
</tr>
<tr>
<td>C</td>
<td>663618</td>
<td>41</td>
<td>153239</td>
<td>561999</td>
</tr>
<tr>
<td>D</td>
<td>663618</td>
<td>41</td>
<td>237799</td>
<td>193582</td>
</tr>
<tr>
<td>ALL</td>
<td>663618</td>
<td>50</td>
<td>172832</td>
<td>799798</td>
</tr>
</tbody>
</table>

- Shared by 3
- Shared by 2

---

\((A \cap B \cap C \cap D) \vee (\neg A \cap B \cap C \cap \neg D)\)
Diff is just an example data source!

- The Variant Analysis model is generic
  - Different system representations possible

- Analysis phases can be adapted to specific needs
  - Different similarity detection algorithms possible
Generalization

Equivalence Relation and Unambiguous Assignment

- Bar diagrams and occurrence matrices can be applied to analyze and visualize any kind of variability
  - Code, non-code artifacts, model elements, features, …

- The prerequisite for using the technique is a “correct” filling of the occurrence matrix

  - **Equivalence relation** across the variants’ elements needed
    - Reflexive $\forall x \in S: \, x \ rel \ x == \ true$
    - Symmetric $\forall x, y \in S: \, x \ rel \ y \Rightarrow y \ rel \ x$
    - Transitive $\forall x, y, z \in S: \, x \ rel \ y \land y \ rel \ z \Rightarrow x \ rel \ z$

  - **Unambiguous assignment** of equivalent elements across variants
    - Necessary if more than one element from variant A is equivalent to a given element of variant B

[S. Duszynski: Visualizing and Analyzing Software Variability with Bar Diagrams and Occurrence Matrices. SPLC 2010]
Limitations

- Typical situation in reverse engineering:
  - Use syntax-level approaches...
  - ... trying to derive meaningful (semantic-level) results
- Variant Analysis retrieves just the syntactic similarity
- It also depends on the structure similarity: comparing non-cloned system does not deliver interesting results
Using the obtained information

Relation to scoping and other information sources

Scoping
- Domain
- Requirements
- Features

Reverse engineering variability
- Similarities and differences
- Structures
- Fine-grained data

Future plans
- Product release schedule
- Products, features to be added or abandoned
- Company strategy

Code quality
- Maintainability
- Bug history
- Stability
- Staff knowledge
Summary

- Occurrence matrices: a data structure to store detailed variability information
- Matrix construction algorithm
  - Scalable: works for any number of variants
  - Generic: supports any element types
  - Flexible: equivalence relations enable customized definitions of similarity

- Bar diagrams: visualization technique for variability information
- Subset calculations: on-demand retrieval of variant intersections

- Generalized framework for analysis of cloned systems
Further work

- Attach a data source more advanced than diff
  - Clone detection results
  - Model-based comparison

- Define further analyses on the rich data set available
  - E.g. variability metrics: granularity, # different configurations needed, ...

- Try to obtain more semantic-level results
  - Mapping features to code, traceability, ...

- Perform (publishable) case studies
Thank you!

Discussion…