Categories of Source Code in Industrial Systems

Tiago L. Alves (SIG & Universidade do Minho)
Software Risk Assessment
- In-depth investigation of software quality and associated business risks
- Answers to specific research questions

Software Monitoring
- Continuous measurement, feedback, and development consultancy
- Guard quality from start to finish

Software Product Certification
- Five levels of technical quality (maintainability)
- Evaluation by SIG, certification by TÜV Informationstechnik
Who is using our services?

- Financial and Insurance companies
  - ABN AMRO
  - ING
  - Rabobank
  - PGGM
  - InterBank
  - Zorg en Zekerheid
  - ochmea
  - Friesland Bank
  - LeasePlan
  - Interpolis
  - Bank Mendes Gans
  - globalcollect
  - Allianz

- Government
  - Rijksoverheid
  - kadaster
  - Confédération Suisse
  - Confederazione Svizzera
  - Belastingdienst
  - Raad voor Rechtsbijstand
  - Politie

- Logistical
  - DHL
  - TNT
  - Getronics PinkRocade
  - Capgemini
  - ProRail
  - Port of Rotterdam

- IT
  - KLM
  - Centric
  - Exact software
  - IBM
  - CHSS
  - Alcatel-Lucent

- Other
  - ENECO
  - essent
  - SWISS LEX
  - KPMG
  - NXP
  - Gasunie
  - Electrabel
Getting the facts straight

Facts (measurements) reported to clients MUST be correct

This applies to any empirical research

DON’T
- Don’t show top 10 highest McCabe methods including generated code
- Don’t measure Fan-in including test code
- Others…
McCabe example
Production + generated code

Tiago L. Alves, ESEM 2011 Short Paper, Banff, AB, Canada. 2011-09-22 © Software Improvement Group
Scoping software analyses

What

• Characterize different source code artifacts
• Account / Remove them in the analysis

How:

1. Semi-automatic methodologies to discover categories
   • Manually look to top-level folder structure
   • Run regular expressions with known keywords
2. Do a technical session with client to validate scoping
1. Propose a categorization

2. Validate categorization with 80 industrial systems (are these categories inhabited and to what degree?)

3. Lessons learned
Source code categories – initial proposal

Source code

Production
- Manually-maintained
- Generated
- Library
- Example

Test
## Experiment results

<table>
<thead>
<tr>
<th></th>
<th>Production</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manually-maintained</td>
<td>80 (100%)</td>
<td>74 (93%)</td>
</tr>
<tr>
<td>Generated</td>
<td>52 (65%)</td>
<td>15 (19%)</td>
</tr>
<tr>
<td>Library</td>
<td>30 (38%)</td>
<td>7 (9%)</td>
</tr>
<tr>
<td>Example</td>
<td>2 (3%)</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Production</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manually-maintained</td>
<td>80 (100%)</td>
<td>56 (70%)</td>
</tr>
<tr>
<td>Generated</td>
<td>34 (43%)</td>
<td>3 (4%)</td>
</tr>
<tr>
<td>Library</td>
<td>6 (8%)</td>
<td>0</td>
</tr>
<tr>
<td>Example</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Representing 5% or more
Empirical evidence
(80 Java and C# Industrial systems)
Categorization issues

Category candidates
• Decompiled code (1 system with 8.25% of code)
• Unused, Temporary and Extraneous code (found in practice)

Categorization assumptions
• Manually-modified generated or library code
• Granularity at file level
Challenges

Short-term

• Formally define scope information (e.g. XML)
• Enable finer/coarser granularity in a language independent manner

Mid-term

• Support software evolution (cope with new, changed and removed artifacts)
• Support validation rules
• Automatic inference and validation of source code categories

Long-term

• Standardization of scope definition (complementary to build definition)
• Support for zero-configuration (static) analysis tools
Conclusion

Proposed initial categorization for software code artifacts

Provided evidence of this categorization analyzing 80 industrial systems

Analyses should take into account different categories

- In average only 60% of the code is “Manually-maintained production” code
- Generated code can represent up to 80% of the overall system code
- Library code can represent up to 40% of the overall system code
Thank you

t.alves@sig.eu