Assignments

- Assignment 6 (Architects and developers)
  - Deadline: Today
    - Some of you got deadline extension till Wednesday

- Assignment 7 (Testing)
  - On Peach
  - Deadline: May 31
    - No class on May 24 (Whit Monday)

- Assignment 8 (Reengineering)
  - To be published on May 31
  - Due to June 21

Sources

- Waterfall model [Royce 1970]

Establishing correctness of the program

- Formal verification
  - Model checking, theorem proving, program analysis
  - Additional artefacts: properties to be established
  - Optional artefacts: models

  Testing
  - Additional artefacts: test cases/scripts/programs
  - Optional artefacts: drivers/stubs

  Co-evolution problem: additional (and optional) artefacts should co-evolve with the production code

Different flavours of tests

<table>
<thead>
<tr>
<th>Testing</th>
<th>Kind of software</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Management IS</td>
</tr>
<tr>
<td>Unit</td>
<td>10</td>
</tr>
<tr>
<td>Integration</td>
<td>5</td>
</tr>
<tr>
<td>System</td>
<td>7</td>
</tr>
<tr>
<td>Acceptance</td>
<td>5</td>
</tr>
</tbody>
</table>

- Effort percentage (staff months) [Capers Jones 2008]
- Evolution research so far focused on unit testing
  - Highest percentage in testing
  - Best-suited for automation
Unit testing

- Test code is also code
  - Recent: unit testing frameworks become popular
- For JUnit code
  - Fixture: common part for multiple tests
    - @Before: set-up, resource claim
    - @After: resource release
  - @Test
  - Traditional metrics can be computed
  - Compare the evolution of the production code metrics and test code metrics

Examples of co-evolution scenarios [Zaidman et al. 2008]

- p – production code
- t – testing code
- Commands – methods with @Test annotation

Co-evolution patterns in Checkstyle

1. Test reinforcement: ↑↑ #test classes
2. Test refinement
3. Intensive development – testing backlog
4. Back to synchronous testing
5. Test refactoring (↓↓ tLOC)

The diagrams seem to suggest

- Correlation between the size of the test suite size and the production code size
  - Reminder: McCabe’s complexity is related to the expected testability effort
  - We are looking at the actual testability effort...

Co-evolution patterns in ArgoUML

1. No correspondence between the production code and the test code: pure testing phase
2. Test skeleton introduction
3. Test refinement
4. Test refactorings

“Initial hill” – changes in the VCS leading to code duplication

The diagrams seem to suggest

- Correlation between the size of the test suite size and the production code size
  - Reminder: McCabe’s complexity is related to the expected testability effort
  - We are looking at the actual testability effort...

- Dependent variables
  - dLOCC – LOC per test class
  - dNOTC – number of test cases
- Independent variables
  - FOUT – Class-out
  - WMC – WMC/McCabe

Quantity vs. Quality

- So far: Quantity (tLOC, tClasses, tCommands)
  - BUT how good are the tests?
- Coverage: measure of test quality
  - % program components “touched” by the tests
  - Variants
    - Statement coverage
    - Function/method coverage
    - Module/class coverage
    - Block coverage
  - Block: sequence of statements with no jumps or jumps’ targets
**Condition coverage vs. Decision coverage**

- **Condition coverage**
  - Every boolean subexpression has been evaluated to true and to false

- **Decision coverage**
  - In every decision (if/loop) both the true and the false branch have been tested

- Does condition coverage imply decision coverage?
- Does decision coverage imply condition coverage?

---

**Path coverage**

- **Path coverage**: all possible paths through the given program
  - Unrealistic: \( n \) decisions \( \Rightarrow \) up to \( 2^n \) different paths
  - Some paths are infeasible
    - Whether a path is infeasible is undecidable

- Coverage implications: path \( \Rightarrow \) decision \( \Rightarrow \) statement

- Special paths: from definition (\( i = 1 \)) to use (\( x += i \))
  - c-use if the use is a computation (\( x += i \))
  - p-use if the use is a predicate (\( x < i \))

---

**The more you test the better the coverage**

- Average over 12 competing versions of the same software
- Coverage increases
- 100% is still a dream even after more than 20,000 tests!

---

**What about evolution of test coverage?**

- High class coverage (>80% and >95% for 4.*)
  - Exception: 2.2
  - 2.*
    - \( pLOC \) increases faster than \( tLOC \)
    - drop in coverage values: major reengineering
  - 3.0-4.0: increase for all forms of coverage

---

**Function coverage in bash**

- Retrospective analysis: tests for version \( i \) were rerun for all versions \( j, j>i \)
- Function coverage
- BUT #functions increases and coverage is percentage
- Consider only functions present in all Bash versions

---

**Condition coverage vs. decision coverage**

```c
int foo(int a, int b) {
    int c = b;
    if ((a>5) && (b>0)) {
        c = a;
    }
    return a*c;
}
```

- \{ foo(-1), foo(5,2) \} covers all conditions but not all decisions
- \{ foo(-1), foo(7,1) \} covers all decisions but not all conditions
Closer look at changes

- Remember eROSE? [Zimmermann et al. 2004]

Association Rule Mining

- eROSE is based on detecting frequent sets and association rules, i.e., elements that often are changed together
  - Popular technique: Apriori algorithm (see Lecture 7)
  - Tests are code, so [Lubsen, M.Sc. thesis]
  - Distinguish tests/production classes based on their names
    - Drop files that are neither source nor test (makefiles, images, etc.)
  - Use Apriori to mine association rules

Quality of rules: A ⇒ B (A, B – sets)

- Support |A∩B| = P(A,B)
- Confidence |A∩B| / |A| = P(B|A)
- Additional values:
  - Lift \(P(A\cap B)/(P(A)\cdot P(B))\)
  - Lift \(= 1\) if A and B are independent
  - High Lift – A and B are “indeed” related
  - Leverage \(P(A,B) - (P(A)\cdot P(B))\)
  - Values > 0 are desirable
  - Conviction \(P(A)\cdot P(\neg B)/P(A,B)\)
  - NB: Asymmetric
  - Strong rule: high confidence (and lift) and reasonable support.

Apriori?

- Support |A∩B| = P(A,B)
- Support of \(X \Rightarrow Y,Z\) equals to support of \(X \Rightarrow Z\) and of \(Y \Rightarrow X,Z\), etc.
- First look for sets with a reasonable support, than for the rules with high confidence (lift)
  - Determine frequent item sets
    - If \(A \cup B\) is frequent, then A and B are frequent
  - Generate association rules from frequent item sets

Apriori: frequent item sets

- \(C_k\): Candidate item set of size \(k\)
- \(L_k\): frequent item set of size \(k\)
- \(L_k = \{\text{frequent items}\}\);
- For (k = 1; \(L_1\)≠∅; k++) do begin
  - \(C_{k+1} = L_k \triangleleft L_k\)
    - Drop those elements that have a non-frequent subset
    - for each transaction do
      - increment the count of all candidates in \(C_{k+1}\) that are contained in \(t\)
    - \(L_{k+1}\) = candidates in \(C_{k+1}\) with minSupport
  - end
- return \(\cup L_k\):

Efficiency bomb

Can be improved...

From sets to rules

- Given frequent item set \(X\)
  - Naïve:
    - For each partition \(Y, Z\) such that \(Y \cup Z = X\)
      - If confidence(\(Y \Rightarrow Z\) < threshold reject;
      - Else report “\(Y \Rightarrow Z\)”
  - However, confidence is \(|Y \cap Z| : |Y|\), so
    - confidence(\(Y \cup Y_2 \Rightarrow Z\) ≥ confidence(\(Y_1 \Rightarrow (Y_2 \cup Z)\))
    - Hence, if confidence(\(Y_1 \cup Y_2 \Rightarrow Z\) < threshold, no need to calculate confidence(\(Y_1 \Rightarrow (Y_2 \cup Z)\))
    - Not all partitions of \(X\) should be considered
    - Start with one-element long \(Z\)...
Rule categorization

- Categorize rules $A \Rightarrow B$ (A, B – classes):
  - PROD: A and B are production classes
  - TEST: A and B are test classes
  - P&T pairs:
    - P2T, T2P
    - mP2T, mT2P: matched pairs ($C_{java} \Rightarrow C_{test.java}$)

- Are there any other types of rules we’ve missed?

Empirical evaluation

<table>
<thead>
<tr>
<th>Rule Class</th>
<th>Checkstyle</th>
<th>A.I</th>
<th>A.II</th>
<th>B.I</th>
<th>B.II</th>
<th>C.I</th>
<th>C.II</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL (N)</td>
<td>10156</td>
<td>14900</td>
<td>8820</td>
<td>210240</td>
<td>27560</td>
<td>499</td>
<td></td>
</tr>
<tr>
<td>PROD</td>
<td>98.85%</td>
<td>35.15%</td>
<td>49.64%</td>
<td>39.06%</td>
<td>99.12%</td>
<td>51.64%</td>
<td>40.56%</td>
</tr>
<tr>
<td>TEST</td>
<td>0.46%</td>
<td>38.11%</td>
<td>9.97%</td>
<td>24.81%</td>
<td>0.36%</td>
<td>9.99%</td>
<td>16.57%</td>
</tr>
<tr>
<td>P&amp;T</td>
<td>0.67%</td>
<td>10.75%</td>
<td>40.23%</td>
<td>36.19%</td>
<td>0.69%</td>
<td>32.44%</td>
<td>32.13%</td>
</tr>
<tr>
<td>P2T</td>
<td>0.33%</td>
<td>15.37%</td>
<td>31.12%</td>
<td>10.18%</td>
<td>0.34%</td>
<td>16.22%</td>
<td>16.06%</td>
</tr>
<tr>
<td>mP2T</td>
<td>0.33%</td>
<td>19.37%</td>
<td>20.12%</td>
<td>18.10%</td>
<td>0.34%</td>
<td>16.22%</td>
<td>16.06%</td>
</tr>
<tr>
<td>T2P</td>
<td>0.33%</td>
<td>19.37%</td>
<td>20.12%</td>
<td>18.10%</td>
<td>0.34%</td>
<td>16.22%</td>
<td>16.06%</td>
</tr>
<tr>
<td>mT2P</td>
<td>0.09%</td>
<td>0.70%</td>
<td>0.74%</td>
<td>0.83%</td>
<td>0.01%</td>
<td>0.78%</td>
<td>4.02%</td>
</tr>
<tr>
<td>UNDEF</td>
<td>0.06%</td>
<td>0.00%</td>
<td>0.18%</td>
<td>0.06%</td>
<td>0.00%</td>
<td>5.31%</td>
<td>10.04%</td>
</tr>
</tbody>
</table>

Empirical evaluation

- A.I, A.II, C.I and C.II (synchronous co-evolution)
  - the ratios correspond to the effort distribution.
  - the confidence of typical rules is not low.

Questions

- Apriori algorithm usually works for A and B as sets of elements rather than individual elements:
  - Age > 52, CurrentAcc = true ⇒ Income > 43759,
    SavingsAcc = true
  - Why did Lubsen consider only pairs of classes?

Assignment 7 (2 students, due to May 31)

- Take a system using JUnit (e.g., Shindig)
- Based on the log-file mine association rules
  - Preprocess the log
  - Miners: Weka, XLMiner, ...
  - Algorithms: Apriori, FP-Growth, ...
  - Parameters: “pairs of classes”, minConfidence, minSupport, ...
- Evaluate the results
  - Distribution over the PROD, TEST, P&T, UNDEF
  - Distribution of quality values (e.g. lift) for each category
  - Visualize: box plot, histogram, kernel density est.
  - Aggregate: Gini/Thiel or fitted distribution

More than JUnit

- There exist JUnit-like systems for
  - Server-side code: Cactus
    http://jakarta.apache.org/cactus/
  - Web-applications: HttpUnit
    http://sourceforge.net/projects/httpunit/
- Popularity?
- No research so far (AFAIK)
Conclusions

- Verification ⇒ Testing ⇒ Unit testing
- Other options are practically unstudied
- Unit testing – another group of code files
  - Traditional metrics are applicable
    - Correlation, co-evolution patterns
  - Coverage metrics
  - Association rules

2IS55 Software Evolution

Implementing evolution: Reengineering

Alexander Serebrenik

Sources

Slides of Rainer Koschke (in German)
http://www.iste.uni-stuttgart.de/ps/Lehre/reengineering/neu/transformationen.pdf

So far...

- We assumed that the evolution has already taken place.
- Remainder of the semester: how to implement evolution
  - Reengineering of legacy systems
    - Towards OO, aspects, services
  - Refactoring and its impact
  - Database migration

Evolution strategies

- Refactor
- Reengineer
  - E.g., using models (see next slides)
  - Re-implent

Questions

- How can one decide which strategy to follow?
- How can/should one implement the chosen strategy?
- What impact does the implementation have on the co-evolving artifacts?

First look at reengineering decision making

Ransom, Sommerville, Warren

Business value

- Reengineering, modernisation
- Replacement

Technical quality

- Reduced maintenance
- Maintenance

- Both technical and business aspects
- Scale is rather vague
Value-Based Decision Model [Visaggio 2000]

- Metrics to assess technical quality and business value

**Examples** | **Business value** | **Technical quality**
--- | --- | ---
**Objective** | • Input volume | • Constants
  • %input that can be automatically processed | • OS calls
  • DB queries/update | • Adaptable

**Subjective (expert opinion)** | • Importance | • Comprehensibility
  • Fitness for purpose | • Correctness
  • Adaptability | • Efficiency

- Each metrics has a threshold B and a weight w (importance)

\[
T_{eq} = \sum \frac{B_i}{m_i} \cdot W_j
\]

Technical quality

Business value

**Empirical validation of VDM [Tilus et al.]**

- Alternative approaches:
  - Maintenance
  - Relative
  - Simplified

  - Agreement: O and ◻ should be replaced, ◇ should be maintained
  - Disagreement: experts always preferred VDM

**Problem with VDM: To the man with a hammer, everything looks like a nail**

- Different problems require different solutions

- Critique table [Aversano et al.]

  - If a problem (left) is detected, consider using technique (up)

**Critique table in practice**

- Two case studies: metrics values and desired thresholds obtained by interviews
- Gap analysis: (metrics – threshold)
- Negative bars are important!

**Selecting all relevant techniques can be too much!**
Language migration can improve interoperability but it also affects many “good” attributes.

**So far: business value and technical quality**

- Whatever decision taken, it should fit the goals of the organization
- Organizational scenarios tool
  - costs: major costs, both financial and nonfinancial
  - benefits: both financial and nonfinancial
  - risks: major sources of risk
  - Automate: replace with IT
  - Infomate: IT augments manual process
  - Transform: IT restructures the process
- SABA: Bennett, Ramage, Munro

**Validation**

<table>
<thead>
<tr>
<th>Validation Method</th>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>SABA</td>
<td>Has been developed gradually, iteration may be used to explore the consequences</td>
<td>OST: empirically tested</td>
</tr>
<tr>
<td>Critique tables</td>
<td>Unknown</td>
<td>Two case studies</td>
</tr>
<tr>
<td>VDM</td>
<td>Longitudinal data gathering during method development</td>
<td>Observations of a real-world renewal project (653 programs)</td>
</tr>
</tbody>
</table>

**Evolution strategy choice: Summary**

- Should take both technical quality and business value into account
- But also customer satisfaction [Sahin, Zahedi]
- Should suggest a specific technique rather than a generic approach
- Should be iterative [Ransom, Sommerville, Warren]

**Scenarios and more**

- Scenarios state what should be achieved
- TST helps to refine archetypal solutions:
  - leave, discard, rebuild, reengineer
  - based on the specifics of the organization assets:
    - software, staff and process
- “Tools” are not really tools but methods