Assignments

• Assignment 6:
  • Architects vs. developers (software metrics)
  • Deadline: May 10
  • 3-4 students

• Questions?
Recap: Software metrics

• So far
  • Metrics scales:
  • Size: LOCs, #files, functionality (function points, API)
  • Complexity: Halstead, McCabe, Henry-Kafura
  • OO:
    – Chidamber-Kemerer (WMC, DIT, etc.)
    – LCOM and variants

• Today
  • Package metrics
    – Aggregation of metrics values
  • Churn metrics
Package metrics

- **Size**: number of classes

- **Dependencies à la fan-in and fan-out**
  - Marchesi’s UML metrics
  - Martin’s $D_n$: abstractness-instability balance or “the normalized distance from the main sequence”
  - PASTA

- **Aggregations of class metrics**
"Fan-out"

PK_1: 5

[Marchesi 1998]

C_e: 1

[Martin 1994]

3

[JDepend]

4

[Martin 2000]
Fan-in

“Fan-in” similarly to the “Fan-out”
• Afferent coupling (Martin)
• PK$_2$ (Marchesi)

Dark: TDD, light: no-TDD
• Test-driven development positively affects $C_a$
  • The lower $C_a$ - the better.
• Exception: JUnit vs. Jericho
  • But Jericho is extremely small (2 packages)

[Hilton 2009]
More fan-in and fan-out

- “Fan-in” similarly to the “Fan-out”
- Afferent coupling (Martin)
- PK₂ (Marchesi)

<table>
<thead>
<tr>
<th>Marchesi</th>
<th>Man-months</th>
<th>#Pack</th>
<th>avg(PK₁)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Railway simulator</td>
<td>13</td>
<td>6</td>
<td>8.7</td>
</tr>
<tr>
<td>Warehouse management</td>
<td>7</td>
<td>5</td>
<td>5.0</td>
</tr>
<tr>
<td>CASE tool</td>
<td>13</td>
<td>5</td>
<td>8.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SAP (Herzig)</th>
<th>Correlation post-release defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afferent</td>
<td>0.091</td>
</tr>
<tr>
<td>Efferent [Martin 2000]</td>
<td>0.157</td>
</tr>
<tr>
<td>Class-in</td>
<td>0.084</td>
</tr>
<tr>
<td>Class-out</td>
<td>0.086</td>
</tr>
<tr>
<td>Fan-in</td>
<td>0.287</td>
</tr>
<tr>
<td>Fan-out</td>
<td>0.148</td>
</tr>
</tbody>
</table>
Evolution of afferent and efferent coupling

- Almost all systems show an increasing trend (Lehman’s growing complexity)
- Project 7 (workflow system) is almost stable but very high!
  - Outsourced development
  - No automated tests
  - Severe maintainability problems

Sato, Goldman, Kon 2007
Package metrics: Stability

Stability is related to the amount of work required to make a change [Martin, 2000].

- **Stable** packages
  - Do not depend upon classes outside
  - Many dependents
  - Should be extensible via inheritance (*abstract*)

- **Instable** packages
  - Depend upon many classes outside
  - No dependents
  - Should *not* be extensible via inheritance (*concrete*)
What does balance mean?

A good real-life package must be **instable** enough in order to be easily modified.

It must be **generic** enough to be adaptable to evolving requirements, either without or with only minimal modifications.

Hence: contradictory criteria.
**Dn – Distance from the main sequence**

**Abstractness =**

\[
\frac{\#\text{AbstrClasses}}{\#\text{Classes}}
\]

**Instability =**

\[
\frac{C_e}{C_e + C_a}
\]

\[
D_n = | \text{Abstractness} + \text{Instability} - 1 |
\]

[ R. Martin 1994 ]

---

Zone of pain

Zone of uselessness

Main sequence

TU/e Technische Universiteit Eindhoven University of Technology
Normalized distance from the main sequence

- Dark: TDD, light: no-TDD
- Test-driven development positively affects $D_n$
  - The lower $D_n$ - the better.
- The same exception (Jericho vs. JUnit)

[Hilton 2009]
Distribution and evolution

Exponential distribution

For all benchmark systems studied, here Vuze 4.0.0.4

Peak: many feature requests (average Dn)
PASTA [Hautus 2002]

• PASTA – Package structure analysis tool

• Dependencies between subpackages

• Some dependencies are worse than others
  • What are the “bad dependencies”?
  • Cyclic dependencies, layering violations
PASTA [Hautus]

- Idea: remove bad (cycle-causing) dependencies
  - **Weight** – number of references from one subpackage to another one.
  - Dependencies to be removed are such that
    - The result is acyclic
    - The total weight of the dependencies removed is minimal
  - Minimal effort required to resolve all the cycles

- Upwards dependencies should be removed
From dependencies to metrics

• PASTA(P) = Total weight of the dependencies to be removed / total weight of the dependencies

• No empirical validation of the metrics

• No studies of the metrics evolution

<table>
<thead>
<tr>
<th>Package</th>
<th>PASTA Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>junit</td>
<td>0%</td>
</tr>
<tr>
<td>org.apache.batik</td>
<td>0%</td>
</tr>
<tr>
<td>org.apache.tools.ant</td>
<td>1%</td>
</tr>
<tr>
<td>java</td>
<td>5%</td>
</tr>
<tr>
<td>org.apache.jmeter</td>
<td>6%</td>
</tr>
<tr>
<td>javax.swing</td>
<td>10%</td>
</tr>
<tr>
<td>org.jboss</td>
<td>11%</td>
</tr>
<tr>
<td>org.gjt.sp.jedit</td>
<td>18%</td>
</tr>
<tr>
<td>java.awt</td>
<td>20%</td>
</tr>
</tbody>
</table>
Metrics for higher-level objects as aggregation of metrics for low-level objects
Aggregation techniques

• **Metrics-independent**
  • Applicable for any metrics to be aggregated
  • Are the results also metrics-independent?
  • Based on econometrics

• **Metrics-dependent**
  • Produces more precise results
  • BUT: needs to be redone for any new metrics
  • Based on fitting probability distributions
Examples of aggregation functions so far

- **Sum**
  - WMC is a class-level metrics defined as the sum of metrics for its lower-level elements (methods)
  - McCabe’s complexity of a file is the sum of McCabe’s complexities of its functions
    - Effort to test all functions of the file
  - Not all metrics are additive (DIT? LCOM???)

- **Average**
  - Maintainability index
  - Central tendency is insufficient for assessment
  - Unreliable for skewed distribution
Metrics independent: Coefficient of variation

- Coefficient of variation: $C = \frac{\sigma}{\mu}$

- Allows to compare distributions with different means

- Sometimes used to assess stability of the metrics
  - Metrics is stable for $C < 0.3$
  - Unreliable for small samples
  - Evolution should be studied…
Metrics are like money

software metrics

How far?

econometric values

• prog. lang.
• domain
• …

• region
• education
• gender
• …
Popular technique: Gini coefficient

- Gini coefficient measure of economic inequality
- Ranges on [0; 1]
- High values indicate high inequality
Gini coefficient: Formally

- Lorenz curve:
  - % of income shared by the lower % of the population

\[
\text{Gini} = \frac{A}{A+B}
\]

- Since \(A+B = 0.5\)
  \[
  \text{Gini} = 2A
  \]
Gini and software metrics [Vasa et al. 2009]

- For most of the metrics on the benchmark systems: $0.45 \leq \text{Gini} \leq 0.75$
- Higher Gini/WMC: presence of generated code or code, structured in a way similar to the generated code (parsers)
Gini and evolution: Spring

- Normally rather stable: programmers accumulate competence and tend to solve similar problems by similar means
<table>
<thead>
<tr>
<th>System</th>
<th>Metrics</th>
<th>Increase</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JabRef</td>
<td>WMC</td>
<td>0.75</td>
<td>Machine generated parser introduced</td>
</tr>
<tr>
<td>Checkstyle</td>
<td>Fan-in (classes)</td>
<td>0.44</td>
<td>Plug-in based architecture introduced.</td>
</tr>
<tr>
<td>Jasper-Reports</td>
<td>#Public methods</td>
<td>0.58</td>
<td>Introduction of a set of new base classes.</td>
</tr>
<tr>
<td>WebWork</td>
<td>Fan-out</td>
<td>0.51</td>
<td>A large utility class and multiple cases of copy-and paste introduced.</td>
</tr>
</tbody>
</table>
Aggregation techniques

• Metrics-independent
  • Applicable for any metrics to be aggregated
  • Are the results also metrics-independent?
  • Based on econometrics

Metrics-dependent

• Produces more precise results
• BUT: needs to be redone for any new metrics
• Based on fitting probability distributions
Metrics-dependent aggregation: Statistical fitting

1. Collect the metrics values for the lower-level elements
2. Present a histogram
3. Fit a (theoretical) probability distribution to describe the sample distribution
   a) Select a family of theoretical distributions
   b) Fit the parameters of the probability distribution
   c) Assess the goodness of fit
4. If a theoretical distribution can be fitted, use the fitted parameters as the aggregated value
Step 1: Histograms

- We have seen quite a number of them already!

Robles et al. 2006: LOC in Debian 2.0 (left) and 3.0 (right)
Histograms are not without problems

• Data: 50 birth weights of children with a severe idiopathic respiratory syndrome

• The same data leads to four different “distributions”

• What can affect the way histogram looks like?
  • Bin width
  • Position of the bin’s edges
Kernel density estimators

- **Advantages**
  - Statistically more sound (no dependency on the endpoints of the bins)
  - Produces smooth curves

- **Disadvantages**
  - Statistically more complex
  - Parameter tuning might be a challenge
Kernel density estimates: Intuition

- Data: -2.1, -1.3, -0.4, 1.9, 5.1, 6.2

Histogram: every value is a rectangle. Shape is a “sum” of the rectangles.

What if each value will be a “bump” that can be added together to create a smooth curve?
Kernel density estimation: Formally

\[ f(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \]

Where

- \( n \) – number of observations
- \( h \) – a smoothing parameter, the “bandwidth”
- \( K \) – a weighting function, the “kernel”

Histogram can be obtained using \( \frac{1}{h} \) as \( K \)

Once \( K \) is chosen one can determine the optimal \( h \).
Histogram vs. Kernel density estimate

Histogram

Kernel density estimate

N = 425   Bandwidth = 0.032

Vuse4004
Step 2: fitting a distribution

- Family of distributions is chosen based on shape
- If the parameters fitting is not good enough try a different one!

Tamai, Nakatani. Negative binomial distribution
Sometimes well-known distributions do not really seem to match.

- **Exponential distribution:**
  \[ f(x) = \lambda e^{-\lambda x} \]

- However, support is \([0;1]\) rather than \([0;\infty)\)!
  - Since \[ \int_{0}^{1} f(x) \, dx = 1 - e^{-\lambda} \]
  - we normalize: \[ g(x) = \frac{f(x)}{\int_{0}^{1} f(x) \, dx} \]

- And use \( \lambda \) to find
Step 3c. Goodness of fit: Pearson $\chi^2$ test

- The test statistic

$$X^2 = \sum_{i=1}^{n} \left( \frac{O_i - E_i}{E_i} \right)^2$$

- O – observed frequency of the result $i$
- E – expected frequency of the result $i$

- Compare $X^2$ with the theoretical $\chi^2$ distribution for the given number of degrees of freedom: $P(\chi^2 > X^2)$
  - Degrees of freedom = number of observations – number of fitted parameters
  - Comparison is done based on table values
  - If the $P(\chi^2 > X^2) < \text{threshold}$ – the fit is good
  - Common thresholds are 0.1, 0.05 and 0.01
Recapitulation: Statistical fitting

1. Collect the metrics values for the lower-level elements

2. Present a histogram

3. Fit a (theoretical) probability distribution to describe the sample distribution
   a) Select a family of theoretical distributions
   b) Fit the parameters of the probability distribution
   c) Assess the goodness of fit

4. If a theoretical distribution can be fitted, use the fitted parameters as the aggregated value
What about the evolution of the aggregated values?

- Geometry library: Jun, subsystem “Geometry”
- Tamai, Nakatani: Negative binomial distribution
  \[ f(x) = \binom{x-1}{k-1} p^k (1 - p)^{x-k} \]
  
  - \( p, k \) – distribution parameters
  - \( \binom{x-1}{k-1} \) - binomial coefficient extended to the reals

- Increase – functionality enhancement
- Decrease – refactoring
Reminder: package metrics

- **Size:** number of classes

- **Dependencies à la fan-in and fan-out**
  - Marchesi’s UML metrics
  - Martin’s $D_n$: abstractness-instability balance or “the normalized distance from the main sequence”
  - PASTA

- **Aggregations of class metrics**
  - Metrics independent: average, sum, Gini coefficient
  - Metrics dependent: Distribution fitting
Measuring change: Churn metrics

- Why? Past evolution to predict future evolution

- Code Churn [Lehman, Belady 1985]:
  - Amount of code change taking place within a software unit over time

- Code Churn metrics [Nagappan, Bell 2005]:

**Absolute:**
- Total LOC, Churned LOC, Deleted LOC, File Count,
- Weeks of Churn, Churn Count, Files Churned

**Relative:**
- M1: Churned LOC / Total LOC
- M2: Deleted LOC / Total LOC
- M3: Files churned / File count
- M4: Churn count / Files churned
- M5: Weeks of churn / File count
- M6: Lines worked on / Weeks of churn
- M7: Churned LOC / Deleted LOC
- M8: Lines worked on / Churn count
Case Study: Windows Server 2003

• Analyze Code Churn between WS2003 and WS2003-SP1 to predict defect density in WS2003-SP1
  • 40 million LOC, 2000 binaries
  • Use absolute and relative churn measures

• Conclusion 1: Absolute measures are no good
  • $R^2 < 0.05$

• Conclusion 2: Relative measures are good!
  • An increase in relative code churn measures is accompanied by an increase in system defect density
  • $R^2 \approx 0.8$
Case Study: Windows Server 2003

- Construct a statistical model
  - Training set: 2/3 of the Windows Set binaries
- Check the quality of the prediction
  - Test set: remaining binaries
- Three models
  - Right: all relative churn metrics are taken into account
Open issues

- To predict bugs from history, but we need a history filled with bugs to do so
  - Ideally, we don’t have such a history

- We would like to learn from previous projects:
  - Can we make predictions without history?
  - How can we leverage knowledge between projects?
  - Are there universal properties?
  - Not just code properties but also properties of the entire software process
Conclusions

• Package metrics
  • Directly defined: $D_n$, Marchesi metrics, PASTA
  • Aggregation based
    − Metrics-independent: average, sum, Gini coefficient
    − Metrics-dependent: fitted distributions

• Churn metrics