Measuring Software Product Quality

Eric Bouwers

June 17, 2014
Software Improvement Group

Who are we?
- Highly specialized advisory company for cost, quality and risks of software
- Independent and therefore able to give objective advice

What do we do?
- Fact-based advice supported by our automated toolset for source code analysis
- Analysis across technologies by use of technology-independent methods

Our mission:
We give you control over your software.
Services

Software Risk Assessment
- In-depth investigation of software quality and risks
- Answers specific research questions

Software Monitoring
- Continuous measurement, feedback, and decision support
- Guard quality from start to finish

Software Product Certification
- Five levels of technical quality
- Evaluation by SIG, certification by TÜV Informationstechnik

Application Portfolio Analyses
- Inventory of structure and quality of application landscape
- Identification of opportunities for portfolio optimization
Measuring Software Product Quality
Some definitions

**Projects**
- Activity to create or modify software products
- Design, Build, Test, Deploy

**Product**
- Any software artifact produced to support business processes
- Either built from scratch, from reusable components, or by customization of “standard” packages

**Portfolio**
- Collection of software products in various phases of lifecycle
- Development, acceptance, operation, selected for decommissioning
Our claim today

“Measuring the quality of software products is key to successful software projects and healthy software portfolios”
Measuring Software Product Quality
The ISO 25010 standard for software quality
Why focus on maintainability?

Initiation: 20% of the costs
Build: 80% of the costs
Acceptation: Maintenance
The sub-characteristics of maintainability in ISO 25010

- Analyze
- Reuse
- Modify
- Test

Modularity
Measuring maintainability

Some requirements

- Simple to understand
- Allow root-cause analyses
- Technology independent
- Easy to compute

Suggestions?
Measuring ISO 25010 maintainability using the SIG model

<table>
<thead>
<tr>
<th></th>
<th>Volume</th>
<th>Duplication</th>
<th>Unit size</th>
<th>Unit complexity</th>
<th>Unit interfacing</th>
<th>Module coupling</th>
<th>Component balance</th>
<th>Component independence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysability</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Modifiability</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Modularity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Reusability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Measuring maintainability

Different levels of measurement

- System
- Component
- Module
- Unit
Source code measurement

Volume

**Lines of code**
- Not comparable between technologies

**Function Point Analysis (FPA)**
- A.J. Albrecht - IBM - 1979
- Counted manually
- Slow, costly, fairly accurate

**Backfiring**
- Capers Jones - 1995
- Convert LOC to FPs
- Based on statistics per technology
- Fast, but limited accuracy

### Table 2. Sample Function Point Calculations

<table>
<thead>
<tr>
<th>Raw Data</th>
<th>Weights</th>
<th>Function Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Input</td>
<td>X 4</td>
<td>4</td>
</tr>
<tr>
<td>1 Output</td>
<td>X 5</td>
<td>5</td>
</tr>
<tr>
<td>1 Inquiry</td>
<td>X 4</td>
<td>4</td>
</tr>
<tr>
<td>1 Data File</td>
<td>X 10</td>
<td>10</td>
</tr>
<tr>
<td>1 Interface</td>
<td>X 7</td>
<td></td>
</tr>
</tbody>
</table>

Unadjusted Total: 30
Complexity Adjustment: None
Adjusted Function Points: 30
Source code measurement

_Duplication_

<table>
<thead>
<tr>
<th>0: abc</th>
<th>34: xxxxx</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: def</td>
<td>35: def</td>
</tr>
<tr>
<td>2: ghi</td>
<td>36: ghi</td>
</tr>
<tr>
<td>3: jkl</td>
<td>37: jkl</td>
</tr>
<tr>
<td>4: mno</td>
<td>38: mno</td>
</tr>
<tr>
<td>5: pqr</td>
<td>39: pqr</td>
</tr>
<tr>
<td>6: stu</td>
<td>40: stu</td>
</tr>
<tr>
<td>7: vwx</td>
<td>41: vwx</td>
</tr>
<tr>
<td>8: yz</td>
<td>42: xxxxx</td>
</tr>
</tbody>
</table>
Source code measurement

*Component balance*

Measure for number and relative size of architectural elements

- CB = SBO × CSU
- SBO = system breakdown optimality, computed as distance from ideal
- CSU = component size uniformity, computed with Gini-coefficient

From measurement to rating
A *benchmark based approach*

Note: example thresholds
But what about the measurements on lower levels?

<table>
<thead>
<tr>
<th>Analysability</th>
<th>Modifiability</th>
<th>Testability</th>
<th>Modularity</th>
<th>Reusability</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Volume**
- **Duplication**
- **Unit size**
- **Unit complexity**
- **Unit interfacing**
- **Module coupling**
- **Component balance**
- **Component independence**

The table above shows the relationships and dependencies between different software metrics. The 'X' indicates a strong correlation or measurement method, while the absence of an 'X' indicates the absence of such a relationship or measurement method.
Source code measurement

Logical complexity


- Academic: number of independent paths per method
- Intuitive: number of decisions made in a method
- Reality: the number of if statements (and while, for, ...)

![Diagram](image)
How can we aggregate this?
Option 1: Summing

<table>
<thead>
<tr>
<th></th>
<th>Crawljax</th>
<th>GOAL</th>
<th>Checkstyle</th>
<th>Springframework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total McCabe</td>
<td>1814</td>
<td>6560</td>
<td>4611</td>
<td>22937</td>
</tr>
<tr>
<td>Total LOC</td>
<td>6972</td>
<td>25312</td>
<td>15994</td>
<td>79474</td>
</tr>
<tr>
<td>Ratio</td>
<td>0.260</td>
<td>0.259</td>
<td>0.288</td>
<td>0.288</td>
</tr>
</tbody>
</table>
Option 2: Average

<table>
<thead>
<tr>
<th></th>
<th>Crawljax</th>
<th>GOAL</th>
<th>Checkstyle</th>
<th>Springframework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average McCabe</td>
<td>1.87</td>
<td>2.45</td>
<td>2.46</td>
<td>1.99</td>
</tr>
</tbody>
</table>

Kent Beck
@KentBeck

characterizing power law distributed data with mean and std deviation is like summarizing moby dick by listing the weights of characters
Option 3: quality profile

<table>
<thead>
<tr>
<th>Cyclomatic complexity</th>
<th>Risk category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 5</td>
<td>Low</td>
</tr>
<tr>
<td>6 - 10</td>
<td>Moderate</td>
</tr>
<tr>
<td>11 - 25</td>
<td>High</td>
</tr>
<tr>
<td>&gt; 25</td>
<td>Very high</td>
</tr>
</tbody>
</table>

**Lines of code per risk category**

- Low: 70%
- Moderate: 12%
- High: 13%
- Very high: 5%

** bénéficarios**

- Springframework
- Checkstyle
- Goal
- Crawjax

**Sum lines of code per category**
First Level Calibration
The formal six step process

1. Metrics extraction
   System → (Entity → Metric × Weight)

2. Weight ratio calculation
   System → (Entity → Metric × WeightRatio)

3. Entity aggregation
   System → (Metric → WeightRatio)

4. System aggregation
   Metric → WeightRatio

5. Weight ratio aggregation
   WeightRatio → Metric

6. Thresholds derivation
   70% 80% 90%
   Metric  Metric  Metric

Legend
→ map relation (one-to-many relationship)
× product (pair of columns or elements)

System
Represents individual systems (e.g. Vuze)

Entity
Represents a measurable entity (e.g. java method)

Metric
Represents a metric value (e.g. McCabe of 5)

Weight
Represents the weight value (e.g. LOC of 10)

WeightRatio
Represents the weight percentage inside of the system (e.g. entity LOC divided by system LOC)

Alves, et. al., Deriving Metric Thresholds from Benchmark Data, ICSM 2010
Visualizing the calculated metrics

Alves, et. al., Deriving Metric Thresholds from Benchmark Data, ICSM 2010
Choosing a weight metric

Alves, et. al., Deriving Metric Thresholds from Benchmark Data, ICSM 2010
Calculate for a benchmark of systems

Alves, et. al., *Deriving Metric Thresholds from Benchmark Data*, ICSM 2010
SIG Maintainability Model

Derivation metric thresholds

1. Measure systems in benchmark
2. Summarize all measurements
3. Derive thresholds that bring out the metric’s variability
4. Round the thresholds

<table>
<thead>
<tr>
<th>Cyclomatic complexity</th>
<th>Risk category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 5</td>
<td>Low</td>
</tr>
<tr>
<td>6 - 10</td>
<td>Moderate</td>
</tr>
<tr>
<td>11 - 25</td>
<td>High</td>
</tr>
<tr>
<td>&gt; 26</td>
<td>Very high</td>
</tr>
</tbody>
</table>

Alves, et. al., Deriving Metric Thresholds from Benchmark Data, ICSM 2010
The quality profile

<table>
<thead>
<tr>
<th>Cyclomatic complexity</th>
<th>Risk category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 5</td>
<td>Low</td>
</tr>
<tr>
<td>6 - 10</td>
<td>Moderate</td>
</tr>
<tr>
<td>11 - 25</td>
<td>High</td>
</tr>
<tr>
<td>&gt; 25</td>
<td>Very high</td>
</tr>
</tbody>
</table>

Sum lines of code per category

<table>
<thead>
<tr>
<th>Lines of code per risk category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
</tr>
<tr>
<td>70 %</td>
</tr>
</tbody>
</table>

Bar charts for Springframework, Checkstyle, Goal, and Crawljax.
Second Level Calibration
How to rank quality profiles?

*Unit Complexity profiles for 20 random systems*

Alves, et. al., *Benchmark-based Aggregation of Metrics to Ratings*, IWSM / Mensura 2011
Ordering by highest-category is not enough!
A better ranking algorithm

```
Require: riskprofiles : (Moderate × High × VeryHigh)*, partition^{N-1}

1: thresholds ← ()
2: ordered[Moderate] ← order(riskprofiles.Moderate)
3: ordered[High] ← order(riskprofiles.High)
4: ordered[VeryHigh] ← order(riskprofiles.VeryHigh)
5: for rating = 1 to (N - 1) do
6:   i ← 0
7:   repeat
8:     i ← i + 1
9:     thresholds[rating][Moderate] ← ordered[Moderate][i]
10:    thresholds[rating][High] ← ordered[High][i]
11:   until distribution(riskprofiles, thresholds[rating]) ≥ partition[rating] or i ≥ length(riskprofiles)
12: index ← i
13: for all risk in (Moderate, High, VeryHigh) do
14:   i ← index
15:   done ← False
16:   while i > 0 and not done do
17:     thresholds.old ← thresholds
18:     i ← i - 1
19:     thresholds[rating][risk] ← ordered[risk][i]
20:   if distribution(riskprofiles, thresholds[rating]) < partition[rating] then
21:     thresholds ← thresholds.old
22:     done ← True
23: end if
24: end while
25: end for
26: end for
27: return thresholds
```

Alves, et. al., Benchmark-based Aggregation of Metrics to Ratings, IWSM / Mensura 2011
Which results in a more natural ordering
## Second level thresholds

### Unit size example

<table>
<thead>
<tr>
<th>Star rating</th>
<th>Low risk ([0, 30])</th>
<th>Moderate risk ([30, 44])</th>
<th>High risk ([44, 74])</th>
<th>Very-high risk ([74, \infty])</th>
</tr>
</thead>
<tbody>
<tr>
<td>★★★★★★</td>
<td>-</td>
<td>19.5</td>
<td>10.9</td>
<td>3.9</td>
</tr>
<tr>
<td>★★★★★☆</td>
<td>-</td>
<td>26.0</td>
<td>15.5</td>
<td>6.5</td>
</tr>
<tr>
<td>★★★★☆☆</td>
<td>-</td>
<td>34.1</td>
<td>22.2</td>
<td>11.0</td>
</tr>
<tr>
<td>★★★☆☆☆</td>
<td>-</td>
<td>45.9</td>
<td>31.4</td>
<td>18.1</td>
</tr>
</tbody>
</table>
SIG Maintainability Model

Mapping quality profiles to ratings

1. Calculate quality profiles for the systems in the benchmark
2. Sort quality profiles
3. Select thresholds based on 5% / 30% / 30% / 30% / 5% distribution

Alves, et. al., Benchmark-based Aggregation of Metrics to Ratings, IWSM / Mensura 2011
Your question ...

Does this work?
SIG Maintainability Model

Empirical validation

- The Influence of Software Maintainability on Issue Handling
  MSc thesis, Technical University Delft
- Indicators of Issue Handling Efficiency and their Relation to Software Maintainability,
  MSc thesis, University of Amsterdam
- Faster Defect Resolution with Higher Technical Quality of Software, SQM 2010
Empirical validation

The life-cycle of an issue

- Issue is reported
- Issue is assigned
- Issue is resolved
- Issue is reopened
- Resolved again
Empirical validation

Defect resolution time

<table>
<thead>
<tr>
<th>Category</th>
<th>Thresholds</th>
<th>Timeframe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>0 - 28 days</td>
<td>4 weeks</td>
</tr>
<tr>
<td>Moderate</td>
<td>28 - 70 days</td>
<td>10 weeks</td>
</tr>
<tr>
<td>High</td>
<td>70 - 182 days</td>
<td>6 months</td>
</tr>
<tr>
<td>Very high</td>
<td>182 days or more</td>
<td></td>
</tr>
</tbody>
</table>

Luijten et al. *Faster Defect Resolution with Higher Technical Quality of Software*, SQM 2010
### Defect resolution vs. 

<table>
<thead>
<tr>
<th></th>
<th>$\rho_s$</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volume</td>
<td>0.29</td>
<td>0.003</td>
</tr>
<tr>
<td>Duplication</td>
<td>0.31</td>
<td>0.002</td>
</tr>
<tr>
<td>Unit size</td>
<td>0.51</td>
<td>0.000</td>
</tr>
<tr>
<td>Unit complexity</td>
<td>0.51</td>
<td>0.000</td>
</tr>
<tr>
<td>Unit interfacing</td>
<td>-0.14</td>
<td>0.897</td>
</tr>
<tr>
<td>Module coupling</td>
<td>0.51</td>
<td>0.000</td>
</tr>
<tr>
<td>Analysability</td>
<td>0.51</td>
<td>0.000</td>
</tr>
<tr>
<td>Changeability</td>
<td>0.64</td>
<td>0.000</td>
</tr>
<tr>
<td>Stability</td>
<td>0.41</td>
<td>0.000</td>
</tr>
<tr>
<td>Testability</td>
<td>0.53</td>
<td>0.000</td>
</tr>
<tr>
<td>Maintainability</td>
<td>0.62</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Luijten et al. *Faster Defect Resolution with Higher Technical Quality of Software*, SQM 2010
Resolution time for defects and enhancements

- Faster issue resolution with higher quality
- Between 2 stars and 4 stars, resolution speed increases by factors 3.5 and 4.0
Higher productivity with higher quality
- Between 2 stars and 4 stars, productivity increases by factor 10

Productivity (resolved issues per developer per month)
Your question ...

Does this work?

Yes

Theoretically
But is it useful?
Software Risk Assessment

Kick-off session

Strategy session

Technical session

Technical validation session

Risk validation session

Final presentation

Final Report

Analysis in SIG Laboratory
Example

Which system to use?
Should we accept delay and cost overrun, or cancel the project?
Example

Very specific questions

<table>
<thead>
<tr>
<th></th>
<th>★★★★★</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volume</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Duplication</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Unit size</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Unit complexity</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Unit interfacing</td>
<td>★★★★☆</td>
</tr>
<tr>
<td>Module coupling</td>
<td>★★★★☆</td>
</tr>
<tr>
<td>Component balance</td>
<td>★★★★☆</td>
</tr>
<tr>
<td>Component independence</td>
<td>★★★☆☆</td>
</tr>
</tbody>
</table>
Software Product Certification

1. system source code
   - Evaluation body

2. evaluation report
   - Certification body

System producer and certification client can be the same organization
Summary

Thank you!
Eric Bouwers
e.bouwers@sig.eu