Myth or Reality?
The Effect of Design Pattern Usage on Software Maintainability

Rudolf Ferenc
University of Szeged, Hungary

PPAP 2013 Keynote
Genova, Italy, March 5, 2013
About me

Rudolf Ferenc

■ Assistant Professor at the Department of Software Engineering, University of Szeged, Hungary
■ PhD in Mathematics and Computer Science in 2005
■ Research areas
  — source code analysis, measurement, quality assurance, change impact analysis, bug detection, design patterns
■ General chair of CSMR 2012
■ Program Co-Chair and Program Committee member at major conferences: CSMR, ICSM, SCAM, ICPC, ESEC/FSE, …
■ Currently leading 5 large EU funded 2-3 years R&D projects
  — 8 industrial partners
  — Topics: Refactoring, Java EE architecture reconstruction, C++ and Java static analysis, Android quality assurance, MAGIC test automation, etc.
Measuring Software Quality

The only valid measurement of code quality: WTFs/minute

(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics
Software Quality Measurement

- Measuring/modeling software quality/maintainability for
  - making the right decisions
  - estimating costs
  - assessing risks

- Theoretical models
  - McCall model
  - Boehm model
  - Dromey model
  - FURPS model

- Standards
  - ISO/IEC 9126 (1,2,3,4) model
  - ISO/IEC 25000 (SQuaRE)
ISO/IEC 9126 & 25000

Functionality
- How easy is it to transfer the software to another environment?
- How reliable is the software?
- Are the required functions available in the software?

Portability
- How easy is it to modify the software?

Maintainability
- How efficient is the software?

Efficiency
- Is the software easy to use?
ISO/IEC 9126 sub-characteristics

Software quality

- Characteristics
  - Functionality
    - Suitability
    - Accuracy
    - Interoperability
    - Security
    - Functionality compliance
  - Reliability
    - Maturity
    - Fault tolerance
    - Recoverability
    - Reliability compliance
  - Usability
    - Understandability
    - Learnability
    - Operability
    - Attractiveness
    - Usability compliance
  - Efficiency
    - Time behaviour
    - Resource Utilization
    - Efficiency compliance
  - Maintainability
    - Analyzability
    - Changeability
    - Stability
    - Testability
    - Maintainability compliance
  - Portability
    - Adaptability
    - Instability
    - Co-existence
    - Replaceability
    - Portability compliance

Metrics
ISO/IEC 9126 in practice

- What low-level metrics to use?
- The standard gives suggestions
  - System plan and specification documentation is necessary
  - Hard to measure them
  - Rarely applicable directly in practice
- Most of the time only the source code is available
  - Adapted models are needed for practical use

Practical models
- Maintainability Index
- SIG Quality Model
- SQALE
- SQUALE
- Quamoco Quality Model
- Columbus Probabilistic Quality Model
Maintainability Index

- An aggregated formula for expressing maintainability
  - \( \text{MI} = 171 - 5.2 \times \ln(V) - 0.23 \times (G) - 16.2 \times \ln(\text{LOC}) \)
    - \( V \) = Halstead Volume
    - \( G \) = Cyclomatic Complexity
    - \( \text{LOC} \) = Lines Of Code

- Problems
  - \( \text{LOC} \) gets the highest weight in the formula
    - The role of \( \text{LOC} \) is questionable
  - The applied constants have no intuitive meanings
  - Traceability problems of the calculated MI value
    - Which metric causes the biggest problem?
  - More complex approaches are needed

- Not ISO/IEC 9126 compatible
SIG model

- Software Improvement Group maintainability model
- Selecting low level metrics and aggregating them using a formula
- Based on ISO/IEC 9126
SIG model
SQALE model

- SONAR implementation
  - Partly ISO/IEC 9126 compatible (new grouping, characteristics)

- Defines low level rules
  - E.g. the comment ratio should be above 25%
  - Every rule violation has a remediation cost

- The ISO/IEC 9126 characteristics and rules are mapped
  - There is an index for every characteristic which is the sum of all remediation cost of its rule violations
  - E.g. SQALE Testability Index (STI)
    - Can be calculated for source code elements
SQALE model

- Rating
  - A, B, C, D, E (A is the best, E worst)
  - Can be calculated for source code elements
    - Determining average development cost (D)
      - E.g. 100 units/KSLOC
    - Summing the remediation costs of the rule violations (R)
    - Calculate the R/D value
      - Rating based on the table
      - E.g. given a class with a size of 5 KSLOC with 10 units of overall remediation cost: 10/500 = 0.02 (2%) -> „D”

- The model calculates the overall cost of conforming the code to the desired rules
- No benchmark

“Measuring programming progress by lines of code is like measuring aircraft building progress by weight.” (Bill Gates)
Problems with many existing models

- Quality is ambiguous, hard to express with a number
  - probability distribution is needed
- Lack of objective reference data
- Other problems
  - Models are often based on a simple linear combination of values
  - Classifying metrics by „magic” formulas or thresholds
  - Weighting dependencies is of limited support
Problems with many existing models

- Let’s ask people, how maintainable a software is…
Columbus QM

- Benchmark of more than 100 Java systems

- Probabilistic approach using “expert voting”

- Based on ISO/IEC 9126
Columbus QM

- Goodness function for every aggregate node
- Aggregating the results to the top -> maintainability
- The result of the model is a goodness function
  - The probabilistic distribution of the system quality
  - An average value can be derived
  - With confidence interval
    - E.g. the quality of the system is between 5 and 7 with 87% probability
- Drill-down of the quality value is possible to class and method-level
  - Relative Maintainability Index for source code elements
## Columbus QM

- Example reports
  - Reporting to developers

<table>
<thead>
<tr>
<th>Source code element</th>
<th>LLOC</th>
<th>McCabe</th>
<th>NUMPAR</th>
<th>CC</th>
<th>PI</th>
<th>NOA</th>
<th>CBO</th>
<th>NOI</th>
<th>NII</th>
<th>NLE</th>
<th>P2</th>
<th>Relative Maintainability Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Source code lines]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>![Source code lines]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>![Source code lines]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>![Source code lines]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Columbus QM

- Example reports
  - Reporting to managers
Design Patterns & Software Quality
Design pattern detection

- Structural analysis / Static analysis
  - Graph pattern matching
  - Constraint resolving
  - Metrics based
  - Similarity matrix based
  - Ontology-based

- Behavioral analysis / Dynamic methods
  - e.g. execution pattern analysis

- Semantic analysis
  - e.g. naming conventions
  - to reduce the false positive rate

- Formal specification/composition analysis
  - to supplement different pattern detection approaches by formally specifying the pattern
Literature Overview

- Design quality and maintainability
  - Effect of design motifs in program comprehension
    - Study: comprehend UML diagrams with and without design motifs
    - Expert and novice software engineers
    - Using an eye tracking device
    - **Result**: patterns play a role in program comprehension and maintenance
      » Yann-Gael Guéhéneuc, Stefan Monnier, Giuliano Antoniol. Evaluating the Use of Design Patterns during Program Comprehension - Experimental Setting
      » Sébastien Jeanmart, Yann-Gael Guéhéneuc. Impact of the Visitor Pattern on Program Comprehension and Maintenance
  - Experiment on the impact of design properties to code quality
    - A suite of metrics for OO design, called MOOD, was adopted
    - Software quality is defined by defect density and rework
    - **Result**: design alternatives may have a strong influence on resulting quality
      » Fernando Brito e Abreu, Walcélio Melo. Evaluating the Impact of Object-Oriented Design on Software Quality
Literature Overview

- Design patterns and maintenance
  - Empirical study on maintenance tasks
    - Groups performed identical maintenance tasks on two different versions - with and without design patterns - of four programs
    - The impact on maintainability was measured by completion time and correctness
    - **Result:** the positive effect of patterns is not clear, depends on pattern type
      - Prechelt, L., Unger, B., Tichy, W., Brössler, P., Votta, L. A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions
      - Vokač, M., Tichy, W., Sjoberg, D.I.K., Arisholm, E., Aldrin, M. A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns - A Replication in a Real Programming Environment
  - **Defect density in pattern vs. non-pattern classes**
    - Some patterns - Singleton, Observer - tend to indicate more complex parts than others, e.g.: Factory.
    - **Result:** Observer and Singleton patterns can serve as indicators of code that requires special attention
      - Vokač, M. Defect Frequency and Design Patterns: an Empirical Study of Industrial Code
Literature Overview

- Design patterns and maintenance
  - Questionnaire based surveys
    - Opinions of experts on how each design pattern helps or hinders them during maintenance
    - Result: design patterns should be used with caution during development because they may actually impede maintenance and evolution
      - Khomh, F., Guéhéneuc, Y.G. Do Design Patterns Impact Software Quality Positively?
      - Yann-Gael Guéhéneuc. Empirical Studies on the Impact of Design Patterns on Quality
      - Foutse Khomh, Yann-Gael Guéhéneuc. Perception and Reality: What are Design Patterns Good For?
Literature Overview

- Design patterns and maintenance
  - Survey on maintenance tasks utilizing design patterns
    - Decomposing maintenance tasks to subtasks and examining the frequency of their use according to the deployed design patterns and whether these patterns are utilized during the change
    - **Result:** performing whichever task while taking existing patterns into consideration yields less faulty code
      - Ng, T.H., Cheung, S.C., Chan, W.K., Yu, Y.T. Do Maintainers Utilize Deployed Design Patterns Effectively?
  - Effect of design pattern comments on maintenance
    - Examined whether explicit design pattern comments help maintenance tasks
      - **Result:** pattern relevant maintenance tasks were completed faster with fewer errors if explicit comments were available
      - Lutz Prechelt, Barbara Unger, Michael Philippsen, Walter Tichy. Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance
Literature Overview

- Anti-patterns and code quality
  - Impact of two anti-patterns on program comprehension
    - Assessed the impact of two anti-patterns on the performance of developers and of their combinations: Blob and Spaghetti Code
    - Result: one anti-pattern does not significantly decrease developers’ performance while the combination of two anti-patterns impedes significantly developers
  - Impact of anti-patterns on class change- and fault-proneness
    - Analyze to what extent classes participating in anti-patterns have higher odds to change or to be subject to fault-fixing than other classes
    - Analyze to what extent these odds (if higher) are due to the sizes of the classes or to the presence of anti-patterns
    - Result: classes participating in anti-patterns are more change and fault-prone than others
      » Foutse Khomh, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, Giuliano Antoniol. An exploratory study of the impact of antipatterns on class change- and fault-proneness
Literature Overview

- General conclusions
  - Are there any?
  - Hard to tell…

- We carried out two small experiments on our own
  - The maintainability of JHotDraw considering design pattern usage
  - The maintainability of the software systems in DBP (the Design Pattern detection tools Benchmark platform)
Experiment #1

The maintainability of JHotDraw considering design pattern usage

Thanks to:
- Peter Hegedus
- Denes Ban
Preparations

- We analyzed all 779 revisions of JHotDraw 7
  - a Java GUI framework for technical and structured graphics
- We calculated the maintainability with our probabilistic software quality model
- We parsed the javadoc tags of the source code for gathering pattern instances, because every design pattern instance is documented in JHotDraw
  - Pattern documentation was introduced in revision 522
  - No false positives!

Basic properties of JHotDraw 7

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
\text{Revision} & \text{Lines of code} & \text{Nr. of packages} & \text{Nr. of classes} & \text{Nr. of methods} & PIn_r & PCl_r \\
\text{(r)} & & & & & & \text{TNCL} \\
\hline
522 & 72472 & 54 & 630 & 6117 & 45 & 11.58\% \\
779 & 81686 & 70 & 685 & 6573 & 54 & 13.28 \% \\
\hline
\end{array}
\]
Tendency of quality attributes…

- …in case of design pattern changes
- Five revisions introduced or removed design pattern instances
  - But in these 5 cases the code changes were always related almost only to pattern implementations

<table>
<thead>
<tr>
<th>Revision (r)</th>
<th>Pattern</th>
<th>Pattern Line Density (PDen.s_r)</th>
<th>Maintainability (M_r)</th>
<th>Testability</th>
<th>Analyzability</th>
<th>Stability</th>
<th>Changeability</th>
</tr>
</thead>
<tbody>
<tr>
<td>531</td>
<td>+3</td>
<td>↑</td>
<td>↑</td>
<td></td>
<td>↑</td>
<td>↑</td>
<td></td>
</tr>
<tr>
<td>574</td>
<td>+1</td>
<td>↑</td>
<td>↑</td>
<td></td>
<td></td>
<td>↑</td>
<td></td>
</tr>
<tr>
<td>609</td>
<td>−1</td>
<td>↓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>716</td>
<td>+1</td>
<td>↓</td>
<td>↑</td>
<td></td>
<td>↑</td>
<td>↑</td>
<td></td>
</tr>
<tr>
<td>758</td>
<td>+1</td>
<td>↑</td>
<td>↑</td>
<td></td>
<td></td>
<td>↑</td>
<td></td>
</tr>
</tbody>
</table>
Tendency of quality attributes...

- 258 revisions
- 0.891 Pearson correlation (p-value < 0.000)
- 0.846 Spearman correlation (p-value < 0.000)
  - Between pattern line density and maintainability
Experiment #2

The maintainability of the software systems in DBP
(the Design Pattern detection tools Benchmark platform)

Thanks to:
- DBP team
- Peter Hegedus
DPB

- Design Pattern detection tools Benchmark platform
- Tools repository
  - DPD Tool 4.5
  - MARPLE-DPD 0.0.20120718.dpd
  - P-MARt
  - Web Of Patterns 1.4.3
- Projects repository
  - JHotDraw 5.1
  - JRefactory 2.6.24
  - JUnit 3.7
  - Lexi 0.1.1 alpha
  - MapperXML 1.9.7
  - Netbeans 1.0.x
  - Nutch 0.4
  - PMD 1.8
  - QuickUML 2001
Preparations

- We dropped out JRefactory and NetBeans
  - Only MARPLE-DPD and P-MARt results were available
- We considered a hit to be true positive if it got at least 3 stars during manual evaluation
- In many cases not all hits were evaluated manually, so we estimated the number of true/false positives based on the distribution of the evaluated ones
- We “normalized” the results by dividing the number of patterns with LOC
- We calculated the maintainability with our probabilistic software quality model
### DPD Tool 4.5

- Pearson & Spearman correlations with QM (maintainability)

<table>
<thead>
<tr>
<th>Project</th>
<th>QM</th>
<th>LOC</th>
<th>all</th>
<th>not eval.</th>
<th>all/LOC</th>
<th>true</th>
<th>true/LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>JHotDraw</td>
<td>6,38</td>
<td>8839</td>
<td>84</td>
<td>48</td>
<td>0,00950</td>
<td>56</td>
<td>0,00634</td>
</tr>
<tr>
<td>JUnit</td>
<td>6,63</td>
<td>4947</td>
<td>18</td>
<td>11</td>
<td>0,00364</td>
<td>13</td>
<td>0,00263</td>
</tr>
<tr>
<td>Lexi</td>
<td>5,16</td>
<td>7097</td>
<td>25</td>
<td>20</td>
<td>0,00352</td>
<td>10</td>
<td>0,00141</td>
</tr>
<tr>
<td>MapperXML</td>
<td>6,86</td>
<td>14866</td>
<td>53</td>
<td>37</td>
<td>0,00357</td>
<td>50</td>
<td>0,00336</td>
</tr>
<tr>
<td>Nutch</td>
<td>3,88</td>
<td>23511</td>
<td>67</td>
<td>54</td>
<td>0,00285</td>
<td>36</td>
<td>0,00153</td>
</tr>
<tr>
<td>PMD</td>
<td>3,16</td>
<td>41470</td>
<td>33</td>
<td>19</td>
<td>0,00080</td>
<td>21</td>
<td>0,00051</td>
</tr>
<tr>
<td>QuickUML</td>
<td>5,59</td>
<td>9207</td>
<td>46</td>
<td>36</td>
<td>0,00500</td>
<td>18</td>
<td>0,00196</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pearson</th>
<th>p-value</th>
<th>Spearman</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation</td>
<td>0,594</td>
<td>0,080</td>
<td>0,679</td>
<td>0,047</td>
</tr>
<tr>
<td></td>
<td>0,696</td>
<td>0,041</td>
<td>0,857</td>
<td>0,007</td>
</tr>
</tbody>
</table>
Pearson & Spearman correlations with QM (maintainability)

<table>
<thead>
<tr>
<th>Project</th>
<th>QM</th>
<th>LOC</th>
<th>all</th>
<th>not eval.</th>
<th>all/LOC</th>
<th>true</th>
<th>true/LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>JHotDraw</td>
<td>6,38</td>
<td>8839</td>
<td>216</td>
<td>157</td>
<td>0,02444</td>
<td>146</td>
<td>0,01652</td>
</tr>
<tr>
<td>JUnit</td>
<td>6,63</td>
<td>4947</td>
<td>94</td>
<td>86</td>
<td>0,01900</td>
<td>24</td>
<td>0,00485</td>
</tr>
<tr>
<td>Lexi</td>
<td>5,16</td>
<td>7097</td>
<td>90</td>
<td>0</td>
<td>0,01268</td>
<td>57</td>
<td>0,00803</td>
</tr>
<tr>
<td>MapperXML</td>
<td>6,86</td>
<td>14866</td>
<td>250</td>
<td>229</td>
<td>0,01682</td>
<td>95</td>
<td>0,00639</td>
</tr>
<tr>
<td>Nutch</td>
<td>3,88</td>
<td>23511</td>
<td>323</td>
<td>310</td>
<td>0,01374</td>
<td>174</td>
<td>0,00740</td>
</tr>
<tr>
<td>PMD</td>
<td>3,16</td>
<td>41470</td>
<td>389</td>
<td>378</td>
<td>0,00938</td>
<td>141</td>
<td>0,00340</td>
</tr>
<tr>
<td>QuickUML</td>
<td>5,59</td>
<td>9207</td>
<td>151</td>
<td>12</td>
<td>0,01640</td>
<td>46</td>
<td>0,00500</td>
</tr>
</tbody>
</table>

Pearson 0,785 0,344
p-value 0,018 0,225
Spearman 0,821 0,143
p-value 0,012 0,380

The manual evaluation of the results of the tool is not complete

- The tool calculates a confidence score which was used to automatically fill the evaluations in the benchmark. A typical comment looks like:
  - “This comment was automatically generated from MARPLE. The manual evaluation of the user was: NOTAVAILABLE. The automatic evaluation from the classifier was 0.5”
- Only a small fraction seems to be really manually evaluated
P-MARt

- Pearson & Spearman correlations with QM (maintainability)

<table>
<thead>
<tr>
<th>Project</th>
<th>QM</th>
<th>LOC</th>
<th>true</th>
<th>true/LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>JHotDraw</td>
<td>6,38</td>
<td>8839</td>
<td>22</td>
<td>0,00249</td>
</tr>
<tr>
<td>JUnit</td>
<td>6,63</td>
<td>4947</td>
<td>8</td>
<td>0,00162</td>
</tr>
<tr>
<td>Lexi</td>
<td>5,16</td>
<td>7097</td>
<td>5</td>
<td>0,00070</td>
</tr>
<tr>
<td>MapperXML</td>
<td>6,86</td>
<td>14866</td>
<td>15</td>
<td>0,00101</td>
</tr>
<tr>
<td>Nutch</td>
<td>3,88</td>
<td>23511</td>
<td>15</td>
<td>0,00064</td>
</tr>
<tr>
<td>PMD</td>
<td>3,16</td>
<td>41470</td>
<td>14</td>
<td>0,00034</td>
</tr>
<tr>
<td>QuickUML</td>
<td>5,59</td>
<td>9207</td>
<td>7</td>
<td>0,00076</td>
</tr>
</tbody>
</table>

Pearson 0,686
p-value 0,044
Spearman 0,857
p-value 0,007
Web Of Patterns 1.4.3

- Pearson & Spearman correlations with QM (maintainability)

<table>
<thead>
<tr>
<th>Project</th>
<th>QM</th>
<th>LOC</th>
<th>all</th>
<th>not eval.</th>
<th>all/LOC</th>
<th>true</th>
<th>true/LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>JHotDraw</td>
<td>6.38</td>
<td>8839</td>
<td>139</td>
<td>117</td>
<td>0.01573</td>
<td>76</td>
<td>0.00860</td>
</tr>
<tr>
<td>JUnit</td>
<td>6.63</td>
<td>4947</td>
<td>42</td>
<td>38</td>
<td>0.00849</td>
<td>21</td>
<td>0.00424</td>
</tr>
<tr>
<td>Lexi</td>
<td>5.16</td>
<td>7097</td>
<td>5</td>
<td>1</td>
<td>0.00070</td>
<td>1</td>
<td>0.00014</td>
</tr>
<tr>
<td>MapperXML</td>
<td>6.86</td>
<td>14866</td>
<td>90</td>
<td>84</td>
<td>0.00605</td>
<td>60</td>
<td>0.00404</td>
</tr>
<tr>
<td>Nutch</td>
<td>3.88</td>
<td>23511</td>
<td>4</td>
<td>1</td>
<td>0.00017</td>
<td>3</td>
<td>0.00013</td>
</tr>
<tr>
<td>PMD</td>
<td>3.16</td>
<td>41470</td>
<td>15</td>
<td>11</td>
<td>0.00036</td>
<td>8</td>
<td>0.00019</td>
</tr>
<tr>
<td>QuickUML</td>
<td>5.59</td>
<td>9207</td>
<td>19</td>
<td>14</td>
<td>0.00206</td>
<td>8</td>
<td>0.00087</td>
</tr>
</tbody>
</table>

Pearson

<table>
<thead>
<tr>
<th>Project</th>
<th>0.704</th>
<th>0.713</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.039</td>
<td>0.036</td>
</tr>
<tr>
<td>Spearman</td>
<td>0.821</td>
<td>0.750</td>
</tr>
<tr>
<td>p-value</td>
<td>0.012</td>
<td>0.026</td>
</tr>
</tbody>
</table>
Correlation of tool results

- Pearson correlation of tool results
  - number of detected pattern instances/system

<table>
<thead>
<tr>
<th></th>
<th>DPD</th>
<th>WOP</th>
<th>P-MARt</th>
<th>MARPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPD Tool Pearson Correlation</td>
<td>1</td>
<td>0.96</td>
<td>0.935</td>
<td>0.69</td>
</tr>
<tr>
<td>p-value</td>
<td>0.000</td>
<td>0.001</td>
<td>0.043</td>
<td></td>
</tr>
<tr>
<td>Web Of Patterns Pearson Correlation</td>
<td>1</td>
<td>0.957</td>
<td>0.671</td>
<td></td>
</tr>
<tr>
<td>p-value</td>
<td>0.000</td>
<td></td>
<td>0.049</td>
<td></td>
</tr>
<tr>
<td>P-MARt Pearson Correlation</td>
<td></td>
<td></td>
<td>1</td>
<td>0.585</td>
</tr>
<tr>
<td>p-value</td>
<td></td>
<td></td>
<td></td>
<td>0.084</td>
</tr>
<tr>
<td>MARPLE-DPD Pearson Correlation</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>p-value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- Myth or reality?
  - The Effect of Design Pattern Usage on Software Maintainability
- No definitive answer yet
- Researchers disagree
- Our small experiments suggest it is reality
  - But much more experiments are needed
Thank you for your attention!

Questions?