Finding and Fixing Design Debt

Rick Kazman
University of Hawaii

My Personal Evolution

SAAM (1994)
ATAM (1998)
CBAM (2001)
My Personal Evolution

When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science.

- William Thompson, Lord Kelvin

My Personal Evolution

There is no sense in being precise when you don't even know what you're talking about.

--John von Neumann
My Focus Today

- Automated SE
- Empirical studies
- Reproducibility
- Industrial relevance
- Usability

In theory, there is no difference between theory and practice. In practice, there is. - Yogi Berra

Finding and Fixing Design Debt
The State of the Practice

• The boat is leaking but you keep paddling!
• Why?
  • The illusion of progress.
  • The lack of measurements.
  • Design is largely invisible.
My "Grand Research Challenge"

• How to measure the health of an architecture?
• Can this be:
  • Automated?
  • Empirically justified?
  • Repeatable?
Isn't This a Solved Problem?

• Just use existing TD detection tools, e.g.

Sadly, no...

• Results of a recent study:
  • TD detection tools disagree about basic (seemingly) objective measures due to different definitions of fundamental concepts.
  • The majority of what is reported by these tools is no more insightful than LOC.

[Lefever et al. ICSE 2021]
There is Hope

- Introducing https://archdia.com/

Empirical Basis

- >300 Open Source Projects
- >50 Industrial projects
DV8 Work Flow

**Step 1: Data Collection**
- Code dependency, history, issue records

**Step 2: Automated Analysis**
- Measurement, hotspot detection, cost calculation

**Step: Collect Feedback**
- Surveys and Interviews with practitioners

---

**DV8 Tool Suite**

**Step 1: Data Collection**
- Source Code
- File Dependency Report
- Revision History
- Bug Report

**Step 2: Automated Analysis**
- 1. SDSM Generator
- 2. HDSM Generator
- 3. DL & PC Calculator
- 4. ArchIssue Detector
- 5. ArchRoot Detector
- 6. ArchIssue Cost Quantification
- 7. Arch Debt Quantification

**Step 3: Collect Feedback**
- Surveys
- Interviews
Step 1: Data Collection

- Dependency information
- History information
- Issue information

Design Rule Space (DRSpace)

A DRSpace is composed of a meaningful subset of a system’s files and the architectural connections among these files.

- Any subset of files may form a design space
- Architectural connections
  - Structural couplings: call, inherit, aggregate, etc.
  - Evolutionary couplings
  - Implicit or explicit

[Xiao et al, TSE 2018]
[Xiao et al, ICSE 2014]
Design Rule Space (DRSpace)

- Non-trivial software system contain multiple design spaces:
  - each feature implemented
  - each pattern applied
  - each concern addressed

- Each file can participate in multiple DRSpaces

- Architectures can be modeled as overlapping DRSpaces

- We visualize each DRSpace as a Design Structure Matrix (DSM)
Step 2: Automated Architecture Analysis

Step 2.1 Measure and Monitor
- Decoupling Level (DL):
  - An options-based metric, measuring the system’s ability to generate options
- Propagation Cost (PC):
  - A DSM-based metric, measuring how tightly coupled a system is

[Mo et al. ICSE 2016]
Decoupling Level (DL): Rationale

A true module should be
- Small
- Independent

A highly modularized system should
- Have large numbers of true modules...
- connected by design rules

[Mo et al. ICSE 2016]

Decoupling Level (DL)

Upper Layer modules:
- The fewer dependents, the higher the value
- The smaller the module, the higher the value

True modules:
- The smaller a true module, the higher the value
- The more true modules, the higher the value

The more files are clustered into true modules, the higher the value

[Mo et al. ICSE 2016]
Decoupling Level (DL) and Propagation Cost (PC)

Data from 129 projects:
- 108 open source
- 21 industrial

[Mo et al. ICSE 2016]
An industrial project:

DL: 29%, 10th percentile: Confirmed to have severe maintenance difficulty

Non-trivial variation in DL indicates major architecture variation

- An industrial project: 6 years, 29 releases, 1082-1852 files
Step 2.2: Flaw Detection

We automatically identify 6 types of design flaws

1. Unstable interface
2. Modularity violation
3. Crossing
4. Improper inheritance
5. Cliques among files
6. Package cycles

These flaws are highly correlated with bugs, changes, and churn

[Mo et al. WICSA 2015]
### Flaw Type 2: Crossing

<table>
<thead>
<tr>
<th>Path</th>
<th>File</th>
<th>Code</th>
<th>Math</th>
<th>Data</th>
<th>Flow</th>
<th>Control</th>
<th>Functionality</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>path1</td>
<td>File1.h</td>
<td>(1)</td>
<td>2</td>
<td>6</td>
<td>11</td>
<td>3</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>path2</td>
<td>File1.h</td>
<td>(2)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>path3</td>
<td>File1.h</td>
<td>(3)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>path4</td>
<td>File1.h</td>
<td>(4)</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>path5</td>
<td>File1.h</td>
<td>(5)</td>
<td>3</td>
<td>2</td>
<td>10</td>
<td>6</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

### Flaw Type 3: Modularity Violation

[Diagram of Flaw Type 3]
Sample Flaws from JBoss

RelationData Manager is in a modularity violation with the command classes.

JDBCStoreManager aggregates command classes and cache class.

Sample Flaws from Cassandra

Cassandra.config.DatabaseDescriptor
Cassandra.utils.FBUtilities
cassandra.config.CFMetaData
Cassandra.dht.RandomPartitioner
Cassandra.utils.CLibrary
Cassandra.io.sstable.SSTable
Cassandra.dht.OrderPreservingPartitioner
Cassandra.locator.PropertyFileSnitch
Do Design Flaws Really Matter?

Research Question: If a file is involved in greater numbers of architecture issues, it is more error-prone/change-prone than average files?

[Mo et al. WICSA 2015]

Data Set

<table>
<thead>
<tr>
<th>Subject</th>
<th>#Month</th>
<th>#Snap</th>
<th>#Comm</th>
<th>#Issue</th>
<th>#File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avro 1.7.6</td>
<td>47</td>
<td>22</td>
<td>1480</td>
<td>630</td>
<td>145-298</td>
</tr>
<tr>
<td>Camel 2.11.1</td>
<td>53</td>
<td>46</td>
<td>17706</td>
<td>2326</td>
<td>528-1203</td>
</tr>
<tr>
<td>Cassandra 1.0.7</td>
<td>24</td>
<td>46</td>
<td>6738</td>
<td>3645</td>
<td>419-786</td>
</tr>
<tr>
<td>CXF 2.7.10</td>
<td>70</td>
<td>92</td>
<td>27247</td>
<td>3400</td>
<td>1426-3073</td>
</tr>
<tr>
<td>HBase 0.94.16</td>
<td>70</td>
<td>21</td>
<td>14858</td>
<td>5032</td>
<td>347-2142</td>
</tr>
<tr>
<td>Ivy 2.3.0</td>
<td>52</td>
<td>11</td>
<td>3799</td>
<td>839</td>
<td>418-607</td>
</tr>
<tr>
<td>OpenJPA 2.2.2</td>
<td>68</td>
<td>17</td>
<td>6736</td>
<td>1574</td>
<td>1216-1761</td>
</tr>
<tr>
<td>PDFBox 1.8.4</td>
<td>46</td>
<td>13</td>
<td>1798</td>
<td>1279</td>
<td>458-589</td>
</tr>
<tr>
<td>Wicket 1.5.5</td>
<td>57</td>
<td>55</td>
<td>18004</td>
<td>3359</td>
<td>1099-1549</td>
</tr>
<tr>
<td>Commercial</td>
<td>9</td>
<td>13</td>
<td>6000</td>
<td>800</td>
<td>137-599</td>
</tr>
</tbody>
</table>
Analysis

We counted the architecture flaws in these 10 projects and compared these to:
• Bug frequency
• Bug churn
• Change frequency
• Change churn

Results

<table>
<thead>
<tr>
<th></th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
<th>#AI</th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
<th>#AI</th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
<td>3.7</td>
<td>0.5</td>
<td>39.0</td>
<td>1</td>
<td>0.5</td>
<td>7.9</td>
<td>2.2</td>
<td>54.2</td>
<td>0</td>
<td>0.4</td>
<td>7.1</td>
<td>1.0</td>
<td>32.6</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
<td>3.9</td>
<td>0.9</td>
<td>26.2</td>
<td>1</td>
<td>1.2</td>
<td>18.5</td>
<td>5.6</td>
<td>131.5</td>
<td>1</td>
<td>1.1</td>
<td>17.4</td>
<td>4.8</td>
<td>106.4</td>
</tr>
<tr>
<td>2</td>
<td>1.6</td>
<td>12.6</td>
<td>5.2</td>
<td>376.7</td>
<td>2</td>
<td>3.7</td>
<td>36.6</td>
<td>14.4</td>
<td>304.7</td>
<td>2</td>
<td>5.3</td>
<td>84.5</td>
<td>21.2</td>
<td>559.1</td>
</tr>
<tr>
<td>3</td>
<td>7.9</td>
<td>124.5</td>
<td>21.6</td>
<td>628.5</td>
<td>3</td>
<td>8.4</td>
<td>341.5</td>
<td>35.9</td>
<td>681.3</td>
<td>3</td>
<td>12.8</td>
<td>245.8</td>
<td>45.7</td>
<td>1202.0</td>
</tr>
<tr>
<td>4</td>
<td>16.5</td>
<td>255.0</td>
<td>33.5</td>
<td>1220.0</td>
<td>4</td>
<td>13.9</td>
<td>204.7</td>
<td>50.9</td>
<td>1043.5</td>
<td>4</td>
<td>18.8</td>
<td>364.9</td>
<td>65.7</td>
<td>1909.4</td>
</tr>
<tr>
<td>PC</td>
<td>0.91</td>
<td>0.89</td>
<td>0.94</td>
<td>0.95</td>
<td></td>
<td>0.96</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
<td>0.9</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
<th>#AI</th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
<th>#AI</th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.8</td>
<td>21.0</td>
<td>2.8</td>
<td>86.9</td>
<td>0</td>
<td>0.4</td>
<td>12.7</td>
<td>1.0</td>
<td>56.8</td>
<td>0</td>
<td>0.7</td>
<td>10.4</td>
<td>0.9</td>
<td>51.0</td>
</tr>
<tr>
<td>1</td>
<td>2.9</td>
<td>62.3</td>
<td>9.4</td>
<td>262.5</td>
<td>1</td>
<td>1.5</td>
<td>24.8</td>
<td>4.2</td>
<td>167.7</td>
<td>1</td>
<td>4.8</td>
<td>236.7</td>
<td>8.3</td>
<td>614.6</td>
</tr>
<tr>
<td>2</td>
<td>8.6</td>
<td>164.8</td>
<td>23.1</td>
<td>592.0</td>
<td>2</td>
<td>5.3</td>
<td>173.6</td>
<td>13.8</td>
<td>558.3</td>
<td>2</td>
<td>9.9</td>
<td>418.5</td>
<td>17.2</td>
<td>2083.6</td>
</tr>
<tr>
<td>3</td>
<td>20.2</td>
<td>360.9</td>
<td>52.5</td>
<td>1232.4</td>
<td>3</td>
<td>28.0</td>
<td>752.1</td>
<td>58.0</td>
<td>1596.6</td>
<td>3</td>
<td>47.8</td>
<td>1355.1</td>
<td>87.6</td>
<td>3158.7</td>
</tr>
<tr>
<td>6</td>
<td>4.1</td>
<td>89.2</td>
<td>142.3</td>
<td>3326.0</td>
<td>4</td>
<td>13.7</td>
<td>237.9</td>
<td>26.8</td>
<td>1252.0</td>
<td>4</td>
<td>76.7</td>
<td>2370.4</td>
<td>135.1</td>
<td>6019.0</td>
</tr>
<tr>
<td>PC</td>
<td>0.90</td>
<td>0.82</td>
<td>0.89</td>
<td>0.89</td>
<td></td>
<td>0.76</td>
<td>0.63</td>
<td>0.72</td>
<td>0.83</td>
<td>0.9</td>
<td>0.94</td>
<td>0.85</td>
<td>0.97</td>
<td>0.97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
<th>#AI</th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
<th>#AI</th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.2</td>
<td>4.5</td>
<td>1.1</td>
<td>31.8</td>
<td>0</td>
<td>1.8</td>
<td>10.0</td>
<td>1.1</td>
<td>36.8</td>
<td>0</td>
<td>0.4</td>
<td>10.1</td>
<td>1.1</td>
<td>92.0</td>
</tr>
<tr>
<td>1</td>
<td>1.1</td>
<td>22.8</td>
<td>3.3</td>
<td>79.6</td>
<td>1</td>
<td>3.2</td>
<td>31.1</td>
<td>3.7</td>
<td>111.5</td>
<td>1</td>
<td>1.4</td>
<td>35.9</td>
<td>2.9</td>
<td>136.5</td>
</tr>
<tr>
<td>2</td>
<td>2.9</td>
<td>54.6</td>
<td>8.4</td>
<td>251.9</td>
<td>2</td>
<td>4.6</td>
<td>68.5</td>
<td>7.5</td>
<td>229.8</td>
<td>2</td>
<td>1.5</td>
<td>66.1</td>
<td>3.4</td>
<td>209.9</td>
</tr>
<tr>
<td>3</td>
<td>7.0</td>
<td>110.9</td>
<td>20.9</td>
<td>666.3</td>
<td>3</td>
<td>10.8</td>
<td>408.6</td>
<td>22.4</td>
<td>862.5</td>
<td>3</td>
<td>8.1</td>
<td>490.0</td>
<td>33.7</td>
<td>861.3</td>
</tr>
<tr>
<td>4</td>
<td>6.4</td>
<td>204.6</td>
<td>18.6</td>
<td>792.3</td>
<td>4</td>
<td>25.1</td>
<td>981.0</td>
<td>52.5</td>
<td>2301.1</td>
<td>4</td>
<td>12.2</td>
<td>689.5</td>
<td>18.4</td>
<td>1254.8</td>
</tr>
<tr>
<td>PC</td>
<td>0.94</td>
<td>0.96</td>
<td>0.97</td>
<td>0.97</td>
<td></td>
<td>0.90</td>
<td>0.88</td>
<td>0.90</td>
<td>0.88</td>
<td>0.9</td>
<td>0.92</td>
<td>0.94</td>
<td>0.94</td>
<td>0.97</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
<th>#AI</th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
<th>#AI</th>
<th>BF avg</th>
<th>BC avg</th>
<th>CF avg</th>
<th>CC avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
<td>2.2</td>
<td>2.7</td>
<td>105.5</td>
<td>0</td>
<td>0.5</td>
<td>6.0</td>
<td>5.3</td>
<td>200.4</td>
<td>0</td>
<td>0.8</td>
<td>5.3</td>
<td>3.7</td>
<td>372.0</td>
</tr>
<tr>
<td>1</td>
<td>0.8</td>
<td>3.2</td>
<td>10.3</td>
<td>372.0</td>
<td>1</td>
<td>2.8</td>
<td>36.8</td>
<td>19.8</td>
<td>864.7</td>
<td>1</td>
<td>6.0</td>
<td>54.0</td>
<td>34.0</td>
<td>1260.0</td>
</tr>
<tr>
<td>PC</td>
<td>0.91</td>
<td>0.93</td>
<td>0.98</td>
<td>0.81</td>
<td></td>
<td>0.90</td>
<td>0.88</td>
<td>0.90</td>
<td>0.88</td>
<td>0.9</td>
<td>0.93</td>
<td>0.94</td>
<td>0.94</td>
<td>0.97</td>
</tr>
</tbody>
</table>
Avro-1.7.6

<table>
<thead>
<tr>
<th>#Flaws</th>
<th>BF_avg</th>
<th>BC_avg</th>
<th>CF_avg</th>
<th>CC_avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
<td>3.7</td>
<td>0.5</td>
<td>29.0</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
<td>3.9</td>
<td>0.9</td>
<td>26.2</td>
</tr>
<tr>
<td>2</td>
<td>1.6</td>
<td>12.6</td>
<td>5.2</td>
<td>376.7</td>
</tr>
<tr>
<td>3</td>
<td>7.9</td>
<td>124.5</td>
<td>21.6</td>
<td>628.5</td>
</tr>
<tr>
<td>4</td>
<td>16.5</td>
<td>255.0</td>
<td>33.5</td>
<td>1220.0</td>
</tr>
<tr>
<td>PC</td>
<td>0.91</td>
<td>0.89</td>
<td>0.94</td>
<td>0.95</td>
</tr>
</tbody>
</table>

More Consequences of Design Flaws

Research Question: If a file is involved in greater numbers of architecture flaws, it is involved in more security bugs/changes than average files?

[Feng et al. WICSA 2016]
We counted the architecture flaws in these 11 projects and compared these to:

- Security bug frequency
- Security change frequency
- ...as well as the original measures (bugs, changes, bug churn, change churn)

<table>
<thead>
<tr>
<th>Project</th>
<th>Bug/Flaw Correlation</th>
<th>Change/Flaw Correlation</th>
<th>Security Bug/Flaw Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avro</td>
<td>0.845</td>
<td>0.923</td>
<td>0.861</td>
</tr>
<tr>
<td>Camel</td>
<td>0.956</td>
<td>0.959</td>
<td>0.958</td>
</tr>
<tr>
<td>Cassandra</td>
<td>0.830</td>
<td>0.869</td>
<td>0.808</td>
</tr>
<tr>
<td>Chrome</td>
<td>0.987</td>
<td>0.988</td>
<td>0.979</td>
</tr>
<tr>
<td>CXF</td>
<td>0.896</td>
<td>0.910</td>
<td>0.939</td>
</tr>
<tr>
<td>Derby</td>
<td>0.938</td>
<td>0.917</td>
<td>0.897</td>
</tr>
<tr>
<td>Hadoop</td>
<td>0.752</td>
<td>0.902</td>
<td>0.862</td>
</tr>
<tr>
<td>HBase</td>
<td>0.894</td>
<td>0.932</td>
<td>0.961</td>
</tr>
<tr>
<td>httpd</td>
<td>0.710</td>
<td>0.688</td>
<td>0.885</td>
</tr>
<tr>
<td>PHP</td>
<td>0.929</td>
<td>0.987</td>
<td>0.923</td>
</tr>
<tr>
<td>Tomcat</td>
<td>0.901</td>
<td>0.776</td>
<td>0.920</td>
</tr>
</tbody>
</table>
Step 2.3: Quantification

- Calculate the costs of each root, each flaw and each type of flaw
- Calculate ROI (Return on Investment)

Industrial Experience: ROI Calculation

<table>
<thead>
<tr>
<th></th>
<th>Penalty Caused by Architecture Debt</th>
<th>Refactoring Cost</th>
<th>Expected Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td><strong>D</strong></td>
<td><strong>E</strong></td>
<td><strong>F</strong></td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>DRSpace Size</td>
<td>Norm Size</td>
<td>Current Defects/Yr</td>
</tr>
<tr>
<td>2</td>
<td>Peer.java</td>
<td>139</td>
<td>119.33</td>
</tr>
<tr>
<td>3</td>
<td>Apple.java</td>
<td>158</td>
<td>133.83</td>
</tr>
<tr>
<td>4</td>
<td>Beon.java</td>
<td>65</td>
<td>37.28</td>
</tr>
<tr>
<td>6</td>
<td>DRSpace Total</td>
<td>290.99</td>
<td>237.8</td>
</tr>
<tr>
<td>7</td>
<td>Project Total</td>
<td>797</td>
<td>265</td>
</tr>
</tbody>
</table>

Result: ~300% ROI in the first year alone!

[Kazman et al. ICSE 2015]
Industrial Experience: Analyzing 8 ABB Projects

- Using 3 complementary techniques:
  - Architecture-level maintainability metrics
  - Architecture flaw analysis
  - Cost and benefit analysis

- 8 projects developed at multiple locations (India, USA, Switzerland) differing in age, domain, and size.

- We reported the results back to each project and collected feedback

[Mo et al. ASE 2018]

Collecting Feedback from ABB

- RQ1: does the tool suite help to close the gap between management and development? That is, does it help them to decide if, when, and where to refactor?

- RQ2: does the tool suite help practitioners understand the maintainability of their systems relative to other projects internal to the company, and relative to a more broad-based benchmark suite?

- RQ3: does the tool suite help developers pinpoint the hotspots of their systems, i.e., the groups of files with severe design flaws?
### ABB: Metrics Scores and Rankings

<table>
<thead>
<tr>
<th></th>
<th>DL</th>
<th>Percentile</th>
<th>PC</th>
<th>Percentile</th>
<th>#Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj_EO</td>
<td>78%</td>
<td>85th</td>
<td>6%</td>
<td>85th</td>
<td>144</td>
</tr>
<tr>
<td>Proj_BM</td>
<td>77%</td>
<td>85th</td>
<td>2%</td>
<td>98th</td>
<td>371</td>
</tr>
<tr>
<td>Proj_CH</td>
<td>76%</td>
<td>81st</td>
<td>16%</td>
<td>54th</td>
<td>6,948</td>
</tr>
<tr>
<td>Proj_EP</td>
<td>72%</td>
<td>74th</td>
<td>7%</td>
<td>83th</td>
<td>1,541</td>
</tr>
<tr>
<td>Proj_SS</td>
<td>57%</td>
<td>49th</td>
<td>20%</td>
<td>45th</td>
<td>15,333</td>
</tr>
<tr>
<td>Proj_OP</td>
<td>57%</td>
<td>49th</td>
<td>21%</td>
<td>41th</td>
<td>7,754</td>
</tr>
<tr>
<td>Proj_CO</td>
<td>55%</td>
<td>43rd</td>
<td>17%</td>
<td>52th</td>
<td>491</td>
</tr>
<tr>
<td><strong>Proj_EC</strong></td>
<td>28%</td>
<td>5th</td>
<td>62%</td>
<td>2nd</td>
<td>4,125</td>
</tr>
</tbody>
</table>

### ABB: Hotspots Detected

- **Unstable Interface**
- **File Cliques**
- **Modularity Violations**
- **Most error-prone files**
ABB: Collecting User feedback

- Surveys
- Interviews

ABB Post-Mortem: Surveys with Architects

Q1: What did the report reveal that you didn’t know about your software?
Q2: Are the metrics useful for reflecting the architecture of your software?
Q3: What did the architecture design flaws reveal about your software?
Q4: What actions have you planned as a result of the architecture design flaws report?
Q5: What did the architecture roots reveal about your software?
Q6: What actions do you plan to take to address architecture roots?
Results

RQ1: does the tool suite help to close the gap between management and development? That is, does it help them to decide if, when, and where to refactor?

✓ Participants of all 8 projects verified that the information provided was useful in closing the understanding gap with management. They have begun the refactoring process.

Results

RQ2: does the tool suite help practitioners understand the maintainability of their systems relative to other projects internal to the company, and relative to a more broad-based benchmark suite?

✓ All participants said the report gave them quantifiable results with which to judge their project. The comparison with industrial benchmarks made it clear that maintenance difficulty caused by degrading architecture is common.
Results

RQ3: does the tool suite help developers pinpoint the hotspots of their systems—that is, the groups of files with severe design flaws?

Six of the eight projects planned to or already started refactoring to address the detected flaws. The project with the lowest DL score is undergoing a major rewrite.

Industrial Experience: Huawei

- Developed a set of architecture measures based on DL and architecture flaws
  - Adopted as a corporate standard
  - Now used in over 100 projects
- Quantified architecture debt
- 24 out of 29 projects studied showed a positive correlation between these measures and productivity

[Wu et al. ECSA 2018]
Industrial Experience: BrightSquid

- Analyzed BrightSquid's secure communication platform (6/16 – 5/17)
- Identified many areas of architecture debt—the "before" state—and recommended a refactoring plan to pay down the debt (7/17)
- Architecture was refactored (1/18 – 3/18)
- Analyzed the "after" state (3/18 – 8/18)

[Nayebi et al. ICSE 2019]

<table>
<thead>
<tr>
<th>General information</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td># of files</td>
<td>1713</td>
<td>711</td>
</tr>
<tr>
<td># of roots covering 80% of bugs</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td># of files in roots covering 80% of bugs</td>
<td>296</td>
<td>295</td>
</tr>
<tr>
<td># of files covering 80% of bugs</td>
<td>17%</td>
<td>37%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectural Metrics</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoupling level</td>
<td>86%</td>
<td>83%</td>
</tr>
<tr>
<td>Propagation cost</td>
<td>6%</td>
<td>6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectural flaws</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td># of cliques</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td># of files influenced by cliques</td>
<td>71</td>
<td>26</td>
</tr>
<tr>
<td># of unhealthy inheritance</td>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td># of files influenced by unhealthy inheritance</td>
<td>222</td>
<td>102</td>
</tr>
<tr>
<td># of unstable interface</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td># of files influenced by unstable interface</td>
<td>471</td>
<td>59</td>
</tr>
<tr>
<td># of crossings</td>
<td>29</td>
<td>6</td>
</tr>
<tr>
<td># of files influenced by crossings</td>
<td>387</td>
<td>47</td>
</tr>
<tr>
<td># of package cycles</td>
<td>34</td>
<td>19</td>
</tr>
<tr>
<td># of files influenced by package cycles</td>
<td>242</td>
<td>94</td>
</tr>
</tbody>
</table>
Industrial Experience: BrightSquid

- The refactoring activities were recorded as 106 change requests, which consumed 563.8 person hours.
- After refactoring, the size of the code base shrunk by 41.5%
- The average time needed to close issues before and after refactoring was reduced by 72%.
- The average bug-fixing churn per issue dropped by 2/3: from 102 LOC before refactoring to 34 LOC after refactoring.
- The average bug-fixing duration reduced 30%, dropping from 10 days before to 7 days.

Lessons Learned

- There is enormous design debt in today's software.
- Yes, in your software.
- That's the bad news.
- The good news: we can do something about it.
Lessons Learned

• The good news: It is possible to automatically and objectively assess and quantify architecture quality – to find and fix the debt.
• And it is possible to bridge the gap. These results were enthusiastically received by the industrial projects.
• Most projects are embarking on major refactorings.
• Several companies have incorporated DV8 into their development processes/pipelines.

Final Thoughts

✔ You can’t manage it if you don’t measure it. Quantification is key.
✔ If the measurement is not automated it won’t be done, or won’t be repeatable.
✔ Incorporating these techniques into the build process ensures rapid feedback with supporting data.
✔ This measurement, detection, and quantification practice leads to improved architectures.
✔ Results must be accompanied by ROI measures, to aid in adoption.

You can get the software—free for academic use—at: https://archdia.com/
Thank You!
Acknowledgments

- I owe a huge debt of gratitude to my co-authors, without whom none of this would be possible.
- Special thanks to:
  - Yuanfang Cai
  - Lu Xiao
  - Ran Mo
  - Qiong Feng
  - Jason Lefever
  - Humberto Cervantes

References
