Architectural Degradation
The Plague of Maturing Software Systems

Nenad Medvidović
Center for Systems and Software Engineering
Computer Science Department
Viterbi School of Engineering
University of Southern California
If you
design software,
implement software,
test software,
maintain software,
manage software projects,
or just use software,
you may be handling a ticking bomb.
Some Terminology

• **Software architecture**
  - Set of principal design decisions \( P \) about a software system

• **Prescriptive architecture**
  - Set of architectural design decisions \( P \) made at time \( t \) that reflect architects’ intent
  - “as designed”

• **Descriptive architecture**
  - Set of artifacts \( A \) that realize the design decisions \( P \)
  - “as implemented”
Two Architectures Side-by-Side

Prescriptive Architecture

Descriptive Architecture
What Happened?

- **Architectural drift**
  - Introduction of design decisions into a system’s descriptive architecture that are not included in, encompassed by, or implied by the prescriptive architecture

- **Architectural erosion**
  - Introduction of design decisions into a system’s descriptive architecture that violate its prescriptive architecture
Why Do We Care?
What about “Real” Examples?

Linux – Prescriptive Architecture

Linux – Descriptive Architecture
Memory Manager Subsystem
Another Example

iRODS – Prescriptive Architecture

iRODS – Descriptive Architecture
And Another

Hadoop Distributed File System – Prescriptive Architecture

HDFS – Descriptive Architecture
Hadoop – Complete Architecture
Hadoop – Complete Architecture
(Somewhat Beautified)
Another Example – bash

Top-Level Architecture
bash – Job Control Component
bash – Commands Component
## How Prevalent Is Architectural Degradation?

<table>
<thead>
<tr>
<th>Technology</th>
<th>PL</th>
<th>KSLOC</th>
<th># Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alchemi</td>
<td>C# (.NET)</td>
<td>26.2</td>
<td>186</td>
</tr>
<tr>
<td>Apache Hadoop</td>
<td>Java, C/C++</td>
<td>66.5</td>
<td>1643</td>
</tr>
<tr>
<td>Apache HBase</td>
<td>Java, Ruby, Thrift</td>
<td>14.1</td>
<td>362</td>
</tr>
<tr>
<td>Condor</td>
<td>Java, C/C++</td>
<td>51.6</td>
<td>962</td>
</tr>
<tr>
<td>DSpace</td>
<td>Java</td>
<td>23.4</td>
<td>217</td>
</tr>
<tr>
<td>Ganglia</td>
<td>C</td>
<td>19.3</td>
<td>22</td>
</tr>
<tr>
<td>GLIDE</td>
<td>Java</td>
<td>2</td>
<td>57</td>
</tr>
<tr>
<td>Globus 4.0 (GT 4.0)</td>
<td>Java, C/C++</td>
<td>2218.7</td>
<td>2522</td>
</tr>
<tr>
<td>Grid Datafarm</td>
<td>Java, C</td>
<td>51.4</td>
<td>220</td>
</tr>
<tr>
<td>Gridbus Broker</td>
<td>Java</td>
<td>30.5</td>
<td>566</td>
</tr>
<tr>
<td>Jcgrid</td>
<td>Java</td>
<td>6.7</td>
<td>150</td>
</tr>
<tr>
<td>OODT</td>
<td>Java</td>
<td>14</td>
<td>320</td>
</tr>
<tr>
<td>Pegasus</td>
<td>Java, C</td>
<td>79</td>
<td>659</td>
</tr>
<tr>
<td>SciFlo</td>
<td>Python</td>
<td>18.5</td>
<td>129</td>
</tr>
<tr>
<td>iRODS</td>
<td>Java, C/C++</td>
<td>84.1</td>
<td>163</td>
</tr>
<tr>
<td>Sun Grid Engine</td>
<td>Java, C/C++</td>
<td>265.1</td>
<td>572</td>
</tr>
<tr>
<td>Unicore</td>
<td>Java</td>
<td>571</td>
<td>3665</td>
</tr>
<tr>
<td>Wings</td>
<td>Java</td>
<td>8.8</td>
<td>97</td>
</tr>
</tbody>
</table>
Reference Architecture for the Grid
Empty Layers – Wings
Skipped Layers – Pegasus
Upcalls – Hadoop
Multi-Layer Components – iRods
A Bit of Everything in Globus

Two layer boundary AND Upcall

Couldn’t determine right “layer”

A Bit of Everything in Globus

Two layer boundary AND Upcall

Couldn’t determine right “layer”

Two layer boundary AND Upcall

Two layer boundary AND Upcall
## Degradation Tally

<table>
<thead>
<tr>
<th>Technology</th>
<th>UC</th>
<th>SL</th>
<th>ML</th>
<th>EL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alchemi</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Apache Hadoop</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Apache HBase</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Condor</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>DSpace</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Ganglia</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GLIDE</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Globus 4.0</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Grid Datafarm</td>
<td>14</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Gridbus Broker</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Jcgrid</td>
<td>4</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OODT</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pegasus</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SciFlo</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>iRODS</td>
<td>36</td>
<td>46</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Sun Grid Engine</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Unicore</td>
<td>4</td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wings</td>
<td>6</td>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Totals:</strong></td>
<td>101</td>
<td>143</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Can We “Smell” Architectural Decay?

• A successful system is maintained over multiple years
• After a while a system becomes like sausage
• When and where do we refactor its architecture?

Architectural smell
  • A commonly made architectural design decision that negatively impacts a system’s lifecycle properties
  • A smell is not a bug; it doesn’t break your system – but that may be the very thing that makes it dangerous
Inspiration

- **Refactoring: Improving the Design of Existing Code**
  by Martin Fowler

- **Code smells** are implementation structures that negatively affect system lifecycle properties

- Defined in terms of *implementation-level* constructs
  - Examples: long parameter list, large methods, ...

- **Code smells do not address architectural decisions**
  - They are not necessarily even correlated with architectural degradation
    [Macia et al. 2012]
A Catalogue of Architectural Smells

- Brick Concern Overload
- Brick Use Overload
- Brick Dependency Cycle
- Unused Interface
- Ambiguous Interface
- Duplicate Component Functionality
- Scattered Functionality
- Component Envy
- Connector Envy
- Connector Chain
- Extraneous Adjacent Connector
A Catalogue of Architectural Smells

- Brick Concern Overload
- Brick Use Overload
- Brick Dependency Cycle
- Unused Interface
- Ambiguous Interface
- Duplicate Component Functionality
- Scattered Functionality
- Component Envy
- Connector Envy
- Connector Chain
- Extraneous Adjacent Connector
Ambiguous Interface – Description

- Three facets of Ambiguous Interface
  - internal dispatch to multiple services
  - offers only one public interface
  - accepts only general type

Must inspect component implementation to know its offered services or its impact
public void onMessage(Message msg) {
    String msgText;
    if (msg instanceof TextMessage) {
        ...
    } else { // If it is not a TextMessage...
        ...
    }
    if (msgText.equalsIgnoreCase("quit")) {
        ...
    }
    ...
    ...
}
Ambiguous Interface – Example

JSP servlet

```java
void service(Request req)
{
    String searchType =
        req.getParameter("searchPref");
    if (searchType.equals("zip")) {
        ...
    } else if (searchType.equals("type")) {
        ...
    } else {
        ...
    }
}
```
Scattered Functionality

- Multiple components are responsible for realizing the same high-level concern
- Some of those components are responsible for orthogonal concerns

- Negatively affects reusability, understandability, testability
Connector Envy

- Components provide interaction-related services
  - Communication
  - Coordination
  - Conversion
  - Facilitation

- Negatively affects reusability, understandability, maintainability
• Connectors of different types link a pair of components

• Connectors’ benefits may cancel each other

➢ Negatively affects reusability, understandability, adaptability
Conclusion

• Your architecture will degrade and get “smelly”
  • Sometimes it starts off that way
• Refactoring is extremely expensive
• You may have no choice
• But how do you know when, where, and how to refactor?
• Understanding and cataloguing smells is only the first step
• We are also working on
  • Improving architectural recovery by exploiting system concerns
  • Identifying smell patterns in systems
  • Understanding their correlation with code smells
Acknowledgments

• **Support from NSF, Bosch RTC**

• **Project participants and collaborators**
  - Yuanfang Cai, Drexel University
  - Chris Douglas, Yahoo Labs
  - Alessandro Garcia, PUC-Rio
  - **Joshua Garcia, USC**
  - Muzzammil Imam, USC
  - **Ivo Krka, USC**
  - **Isela Macia, PUC-Rio**
  - Chris Mattmann, NASA/JPL
  - Daniel Popescu, Google
  - Arndt von Staa, PUC-Rio