Why Developers Overlook Architecture Degradation Symptoms?

Alessandro Garcia

SEIF Workshop 2013
Rio de Janeiro

LES | DI | PUC-Rio - Brazil
Software has an “architecture” too!
How to achieve good architectures?

Keep it simple!

Component addresses a single concern

Loosely coupled components

Simple interfaces
It does not matter...

... if the intended architecture is well defined:

but...
... but the program is not compliant to it!

The actual architecture is in the source code:
Software Architecture Degradation is...

- continuous quality decay of architecture design
- evolving systems: changes are made everyday
... Why do we care?
Memory manager component - Linux
Why do we care?

Actual architecture - iRODS

violations of the intended architecture

Why do we care?

- Hadoop
Why do we care?

Actual architecture - Hadoop

Intended architecture of a software system

- defines how developers actually communicate and work on the system’s “building blocks”
Actual architecture of the same...

... software system
Architectural Erosion

Intended architecture

Actual architecture in the program
Architectural Erosion

Intended architecture

Actual architecture in the program
Architectural Drift

Intended architecture

No dependency violation!

Scattered parasitic functionality

Connector envy

Ambiguous interface

Concern overload

Unused interface

Bloated interface
Drift often manifest as code anomalies...

... God Classes,
Feature Envies, ....
... and many other program anomalies
Why do we care?

- Netscape, Mozilla, EJB, FindBugs and ArgoUML
  - several years of production
- These projects involved US$ millions
  - ... millions and millions of users in many countries
  - ... dozens of developers
- Degradation affects several software domains:
  - Health care, mobile applications, banking, financial market analysis, ...
Recent Advances...

- **Architecture recovery** techniques
  - Recovery of actual architectures from source code

- **Drift detection** in actual architectures
  - Metrics-based strategies for programs

- **Erosion detection**
  - Use of DSLs to describe and check architecture rules
Architecture Recovery techniques are...

- ... useless to support detection of architectural problems in the program in these cases
  - they retrieve components, which do not correspond to actual components
    - reason: intended software architecture is already diffused; packages do not match architectural components
  - they do not retrieve enough information: interfaces, dependencies, etc...
    - reason: intended software architecture is already diffused
Recent Advances...

- Architecture recovery techniques
  - Recovery of actual architectures from source code

- Erosion detection in source code
  - DSLs to describe (and check) anti-erosion rules

- Drift detection in source code
  - DSLs to describe (and check) anti-drift rules
  - Metrics-based strategies
Existing Anti-Degradation Techniques

... supports either drift- **OR** erosion-prevention rules
... for different program languages
Anti-Erosion and Anti-Drift Rules

**Architectural Mapping**

* ArchitecturalConcept Action { parent AbsAppAction}  
  ArchitecturalConcept Engine { suffix Engine }
  ...

**Anti-Erosion Rules**

* only Action can-access Engine  
  Action must declare Services  
  ExportCSV must access "javax.servlet.http"  
  ExportCSV cannot access Indicators, Layout  
  Engine must depend Indicators, Layout

**Anti-Drift Rules**

* GodClass {  
  Coupling > 7  
  Cohesion, TopValues(25%)  
  MethodComplexity, TopValues(25%)  
  }

Bloated Interfaces, Ambiguous Interfaces
Why Developers Overlook ...

... Architecture Degradation Symptoms?
Why Developers Overlook ...

... Architecture Degradation Symptoms?

Detection Accuracy

Detection Effort
Empirical Methods

- Exploratory quantitative studies
  - 7 software projects, such as:
    - PDP – Company X
    - Platform for financial market analysis – Company Y
    - OODT – NASA/Apache
    - MIDAS – Bosch
    - Logistics Framework – Company Z

- Case studies (*in situ*)
  - 7 software projects in the same domain
  - Observations, questionnaires and interviews with architects and developers
Why Developers Overlook ...

... Architecture Degradation Symptoms?

7 Lessons Learned
Why Developers Overlook …

… Architecture Degradation Symptoms?

Detection Accuracy

Detection Effort
Downstream Analysis

Architecture problems and code anomalies were related in

>80%
Upstream Analysis

Lack of ranking support

<table>
<thead>
<tr>
<th>File Name</th>
<th>Number of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>M.Prepare</td>
<td>34</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>34</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>33</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>31</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>30</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>29</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>25</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>20</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>19</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>16</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>16</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>16</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>12</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>12</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>11</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>11</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>11</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>9</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>9</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>8</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>8</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>8</td>
</tr>
<tr>
<td>M.Prepare</td>
<td>8</td>
</tr>
<tr>
<td>M.File</td>
<td>7</td>
</tr>
<tr>
<td>M.File</td>
<td>7</td>
</tr>
<tr>
<td>M.File</td>
<td>7</td>
</tr>
<tr>
<td>M.File</td>
<td>7</td>
</tr>
<tr>
<td>M.File</td>
<td>6</td>
</tr>
<tr>
<td>M.File</td>
<td>6</td>
</tr>
<tr>
<td>M.File</td>
<td>6</td>
</tr>
<tr>
<td>M.File</td>
<td>6</td>
</tr>
<tr>
<td>M.File</td>
<td>5</td>
</tr>
<tr>
<td>M.File</td>
<td>5</td>
</tr>
<tr>
<td>M.File</td>
<td>5</td>
</tr>
<tr>
<td>M.File</td>
<td>5</td>
</tr>
<tr>
<td>M.File</td>
<td>5</td>
</tr>
<tr>
<td>M.File</td>
<td>5</td>
</tr>
<tr>
<td>M.File</td>
<td>5</td>
</tr>
<tr>
<td>M.File</td>
<td>5</td>
</tr>
<tr>
<td>M.File</td>
<td>4</td>
</tr>
<tr>
<td>M.File</td>
<td>4</td>
</tr>
</tbody>
</table>
Too many **DRIFT** candidates to inspect...

... detect thousands of code anomalies
But many irrelevant code anomalies

Up to 80% of analyzed code anomalies were NOT the cause of architecture problems
Architectural Relevance of Code Anomalies

public class HWFacade{
    public void updateComplaint(...){..}
    public Complaint searchComplaint(...){..}
    public void insertComplaint(...){..}
    public void insertEmployee(...){..}
    public Employee searchEmployee(...){..}
    public void updateEmployee(...){..}
    public void insertSymptom(...){..}
    public Symptom searchSymptom(...){..}
    public void updateSymptom(...){..}
    ...
}

public class ComplaintRepo{
    ...
    public int insert(...){..}
    public void update(...){..}
    public int getIndex(...){..}
    public boolean exists(...){..}
    public Complaint search(...){..}
    public void reset(...){..}
    public Object next(...){..}
    public void remove(...){..}
    public List getList(...){..}
    public boolean hasNext(...){..}
    public void updateTimestamp(...){..}
    public int searchTimestamp(...){..}
    ...
}

public class ComplaintRepo{
    ...
    public int insert(...){..}
    public void update(...){..}
    public int getIndex(...){..}
    public boolean exists(...){..}
    public Complaint search(...){..}
    public void reset(...){..}
    public Object next(...){..}
    public void remove(...){..}
    public List getList(...){..}
    public boolean hasNext(...){..}
    public void updateTimestamp(...){..}
    public int searchTimestamp(...){..}
    ...
}

ARCHITECTURE

GUI
Business
DATA
EmployeeArray
ComplaintRepo
Repository
Factory
7 Lessons – Why Developers Overlook ... 

... Architecture Degradation Symptoms?

1. Lack of prioritization support
Studying prioritization models

- Which other characteristics could be explored for detecting architecturally-relevant code anomalies?
  - Change density
  - Error density
  - Anomaly density
  - Code anomaly type
  - Etc...


## Prioritization heuristics

<table>
<thead>
<tr>
<th>System</th>
<th># of Ranked CE</th>
<th>Arch. Relevant</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>14</td>
<td>10</td>
<td>71%</td>
</tr>
<tr>
<td>MM</td>
<td>10</td>
<td>7</td>
<td>70%</td>
</tr>
<tr>
<td>PDP</td>
<td>10</td>
<td>10</td>
<td>100%</td>
</tr>
</tbody>
</table>

### Change density

<table>
<thead>
<tr>
<th>System</th>
<th># of Ranked CE</th>
<th>Arch. Relevant</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>14</td>
<td>12</td>
<td>85%</td>
</tr>
<tr>
<td>MM</td>
<td>10</td>
<td>8</td>
<td>70%</td>
</tr>
<tr>
<td>PDP</td>
<td>10</td>
<td>8</td>
<td>70%</td>
</tr>
<tr>
<td>MIDAS</td>
<td>10</td>
<td>6</td>
<td>90%</td>
</tr>
</tbody>
</table>

### Error density

<table>
<thead>
<tr>
<th>System</th>
<th># of Ranked CE</th>
<th>Arch. Relevant</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>10</td>
<td>7</td>
<td>60%</td>
</tr>
<tr>
<td>MM</td>
<td>10</td>
<td>9</td>
<td>70%</td>
</tr>
<tr>
<td>PDP</td>
<td>10</td>
<td>8</td>
<td>70%</td>
</tr>
</tbody>
</table>

### Anomaly density
7 Lessons – Why Developers Overlook …

… Architecture Degradation Symptoms?

1. Lack of prioritization support
2. There is no ‘universal’ prioritization model
3. Prioritization models: satisfactory results too late
## Prioritization heuristics

<table>
<thead>
<tr>
<th>System</th>
<th># of Ranked CE</th>
<th>Arch. Relevant</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>14</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>10</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>PDP</td>
<td>10</td>
<td>10</td>
<td>100%</td>
</tr>
</tbody>
</table>

### Change density

<table>
<thead>
<tr>
<th>System</th>
<th># of Ranked CE</th>
<th>Arch. Relevant</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>14</td>
<td>12</td>
<td>85%</td>
</tr>
<tr>
<td>MM</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>PDP</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>MIDAS</td>
<td>10</td>
<td>6</td>
<td>90%</td>
</tr>
</tbody>
</table>

### Error density

<table>
<thead>
<tr>
<th>System</th>
<th># of Ranked CE</th>
<th>Arch. Relevant</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>10</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>PDP</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

### Anomaly density

<table>
<thead>
<tr>
<th>System</th>
<th># of Ranked CE</th>
<th>Arch. Relevant</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>10</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>PDP</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>MIDAS</td>
<td>10</td>
<td>6</td>
<td>90%</td>
</tr>
</tbody>
</table>
Earliness of Anomaly

- Early anomalies often appear in the 1st version

18%

Of all architecturally-relevant code anomalies were identified as early anomalies
Earliness of Architectural Problems

- Early anomalies often appear in the 1st version

18% Of all architecturally-relevant code anomalies were introduced as early

Related to almost 40% of all the architectural problems
Example

1\textsuperscript{st} version

```java
public class HWFacade{

    public void updateComplaint(..){..}
    public Complaint searchComplaint(..){..}
    public void insertComplaint(..){..}

    public void insertEmployee(..){..}
    public Employee searchEmployee(..){..}
    public void updateEmployee(..){..}

    ...
}
```

CBC = 7
Example: fixing here is expensive

Version 10

```java
public class HWFacade{

public void updateComplaint(..){..}
public Complaint searchComplaint(..){..}
public void insertComplaint(..){..}

public void insertEmployee(..){..}
public Employee searchEmployee(..){..}
public void updateEmployee(..){..}

public void insertSymptom(..){..}
public Symptom searchSymptom(..){..}
public void updateSymptom(..){..}

... }
```

![Diagram of HWFacade class with methods highlighted as relevant.](image-url)
Priorization Heuristics: conclusions

- Heuristics proposed were able to correctly outline architecturally-relevant anomalies
  - Ranked elements were architecturally relevant in 75%-85% average
- Anomaly density heuristic presented very good results
  - Code modules infected by multiple code anomalies were often related to architectural problems
  - Identification of code anomaly patterns
- Mapping-based prioritization was even better
  - ... but there is a cost involved to produce and maintain these architecture-code mappings
7 Lessons – Why Developers Overlook ... 

... Architecture Degradation Symptoms?

1. Lack of prioritization support
2. There is no ‘universal’ prioritization model
3. Prioritization: satisfactory results too late
4. Critical code anomalies are often introduced early
What about Upfront Detection?

... when developers write their own architectural rules?

- **2nd stage** - Case studies *(in situ)*: 7 software projects

  Observations, questionnaires and interviews
Empirical Methods

- **1st Stage** - Exploratory quantitative studies
  - 7 software projects, such as:
    - PDP – Radix Engenharia
    - Platform for financial market analysis – Minds@Work
    - OODT – NASA/Apache
    - MIDAS – Bosch
    - Logistics Framework – Petrobras/PUC-Rio

- Case studies (*in situ*)
  - 1 case study: accuracy vs. effort
  - 6 software projects in the same domain: reuse of rules
  - Observations, questionnaires and interviews with architects and developers
Why Developers Overlook …

… Architecture Degradation Symptoms?

Detection Accuracy

Overall Effort

- Architectural Mapping
- Rule Description
- Architecture Problem Detection
What about Upfront Detection?

- Exploratory quantitative studies of 7 software projects, such as:
  - PDP – Radix Engenharia
  - Platform for financial market analysis – Minds@Work
  - OODT – NASA/Apache
  - MIDAS – Bosch
  - Logistics Framework – Petrobras/PUC-Rio

Comparison:

Specification and Detection of Architectural Rules vs. Code Inspection
<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural Rules</td>
<td>85%.. 95%</td>
<td></td>
</tr>
<tr>
<td>(... but a few ‘universal’ drift</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rules could be reused)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Inspection</td>
<td>85%.. 95%</td>
<td></td>
</tr>
</tbody>
</table>
False positives were related to ... 

...the nature of multiparadigm of software projects
What about Upfront Detection?

**Accuracy**

<table>
<thead>
<tr>
<th>Architectural Rules</th>
<th>90%.. 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Inspection</td>
<td>90%.. 100%</td>
</tr>
</tbody>
</table>

**Overall Effort**

(Per subsystem)

- **22 hours**
- **16 hours**

-37.5%
Effort is Too High

**Configuration Effort**
(per subsystem)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Configuration (hour)</th>
<th>Detection (hour)</th>
<th>Total (hour)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code inspection</td>
<td>0</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Architectural rules</td>
<td><strong>20</strong></td>
<td>2</td>
<td><strong>22</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Architectural Mapping (hour)</th>
<th>Rules Tailoring (hour)</th>
<th>Total (hour)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural rules</td>
<td><strong>12</strong></td>
<td><strong>8</strong></td>
<td><strong>20</strong></td>
</tr>
</tbody>
</table>

(anti-drift rules)
Reuse to pay off the upfront effort?

- Reuse of architectural rules

Family of 7 Systems
Same ‘Architecture Reference’
Same Company, Practices, Frameworks

Same Domain: Financial Market Analysis
7 Lessons – Why Developers Overlook ... 

... Architecture Degradation Symptoms?

1. Lack of prioritization support
2. There is no ‘universal’ prioritization model
3. Prioritization models tend to yield satisfactory results too late
4. Critical code anomalies are often introduced early
5. Effort on upfront detection is costly or prohibitive
6. False negatives in multi-paradigm software projects
7. Reuse of anti-drift rules are hard
Possible solutions

- Better support for reuse of architectural rules
  - Per concerns in a domain
  - Our initial results are promising
- Synthesizing code anomalies -> architectural problems
- Further study degradation symptoms in multi-paradigm projects
- Exploit informal architectural blueprints to improve static analysis and early detection
The Opus Team and Collaborators

Examples of Collaborators...

USC
USA

TU Darmstadt
Germany
Why Developers Overlook Architecture Degradation Symptoms?

Alessandro Garcia

SEIF Workshop 2013
Rio de Janeiro

LES | DI | PUC-Rio - Brazil