ABSTRACT

Testing is an extremely important part of the software lifecycle, accounting for as much as 50% of the total development costs for many projects. Formalised testing is no longer the exclusive domain of safety critical applications, as toleration of faults and bugs in software is reducing across the software world. With this in mind, rigorous tools, methodologies and solutions are needed to automate as much of the testing work as possible.

One problem often encountered in testing large, monolithic applications is the lack of clarity in the architecture of a software application. Often, a lack of process regarding the architecture leads to so-called “spaghetti code”, where parts of the system are characterised by inappropriate relationships, unstructured code blocks and “blobs”. Such systems are often impossible to test. Without formal architectural rules in place, and a system or tool to enforce these rules and design intent, the system often becomes unmanageable and untestable. Such systems are also prone to documentation problems, change control issues and generally poor lifecycle management.

This presentation first of all examines the problem of specifying architecture and communicating design intent formally, and then secondly, how to test the implementation of this design intent. It is aimed at software architects, designers, developers and testers. Using advanced DSM-based techniques and tools, it is now possible for the developer and architect to work together to ensure that the problem of architectural degradation and erosion of a software system, so often the biggest problem a software project faces, remains a thing of the past. Architects, developers and testers can now work together to produce better software that is also easier to extend, reuse and understand.

Many software problems are introduced when the design is converted to implementation during the coding phase of a project. The benefits of an architecture-first testing approach are shown through a practical demonstration. It can also be shown that a clean architecture leads to reductions in time taken for unit and integration testing phases to complete. At the developer desktop level, a clean architecture can help identify code coverage / dead code issues, and thus many defects can be found even earlier in the lifecycle by adopting this approach.

Contact: Neil Langmead
Emenda Software Limited
Elisabethstrasse 91
80797 Munich
Germany
Tel.: +49 (0) 89 59 08 - 2029
Fax: +49 (0) 89 59 08 - 1200
Mobile: +49 (0) 173 691 3617
Web: www.emenda.eu
Testing the Software Architecture

Neil Langmead
Emenda

What you will learn today

- A new approach to specify and test software architecture by utilizing inter-module dependencies.
- How knowledge of the architecture can help improve your testing

This session includes an actual demo and several real life examples
The Problem - How uncontrolled software complexity happens

<table>
<thead>
<tr>
<th>Plan</th>
<th>Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Tight design spec&lt;br&gt;- Solid architecture&lt;br&gt;- Comprehensive Functionality</td>
<td>- Reduced functionality&lt;br&gt;- Architecture erosion&lt;br&gt;- Limited testing to meet ship date&lt;br&gt;- Plan to, “fix bugs in v1.5.”</td>
</tr>
<tr>
<td>Plan</td>
<td>Reality</td>
</tr>
<tr>
<td>- Clean up bugs&lt;br&gt;- Take app to “next level”&lt;br&gt;- Respond to 1st customer fixes&lt;br&gt;- Control Software Complexity</td>
<td>- Key architects &amp; developers gone&lt;br&gt;- Documentation is non-existent&lt;br&gt;- New team on learning curve&lt;br&gt;- Security concerns</td>
</tr>
<tr>
<td>Plan</td>
<td>Reality</td>
</tr>
<tr>
<td>- Must fix the architecture&lt;br&gt;- No new defects&lt;br&gt;- Reuse components for other applications&lt;br&gt;- Outsource some development</td>
<td>- Complexity and code is growing too quickly&lt;br&gt;- Bug fix versus feature battle underway&lt;br&gt;- Architecture is unknown</td>
</tr>
</tbody>
</table>
How do we efficiently test & add features to this software?

... what software looks like after business success!

Testing the Architecture with Design Rules

- Succinct specification of acceptable and unacceptable dependencies between subsystems
- Each cell of the DSM represents design intent
- DSM offers a powerful way to visualize and specify design rules
- Design Rules enable testing of architecture

Dependency Model = DSM + Design Rules
Design Rules

External rules control library and 3rd party usage
Internal rules are set at highest level of hierarchy and inherited down to lowest level

Rules for Layering
1. $root can-use $root
2. model cannot-use application
3. domain cannot-use application, model
4. framework cannot use application, model, domain
5. util cannot-use application, model, domain, framework
Example: ANT Conceptual Architecture

Layered Architecture with three subsystems
Tasks use common infrastructure

**Key Goal:** Allow independent development of tasks
Improve Testing with Architectural Knowledge

Architectural Visibility improves Testing

Design Rules communicate the intent and enable testing
Conceptual Architecture of Eclipse?

Dependency Model View of Eclipse Platform
Precise big picture view derived from the Dependencies View shows accurate layering and vertical splitting.

Testing Data Architecture …
Testing the Enterprise Architecture

Summary: Testing the Big Picture View

- Specify the big picture view using DSMs - approach allows you to represent massive systems
- Formalize design intent so you can test architectural erosion
- Easy to adopt—Use it at any stage of the lifecycle
- Use the knowledge of the architecture to improve your testing